UNIVERSIDADE DE LISBOA

FACULDADE DE CIÊNCIAS

DEPARTAMENTO DE INFORMÁTICA

Solução Mobile – Augmented Reality & Internet of Things integration

Pedro Duarte Vieira Gomes

Mestrado em Engenharia Informática
Especialização em Engenharia de Software

Trabalho de Projeto orientado por:
Prof. Doutor Naercio David Pedro Magaia

2019

Agradecimentos

Agradeço aos meus pais pelos ensinamentos que me transmitiram, por todo o apoio que me deram e por estarem sempre presentes na minha vida. Sem eles nada do que consegui alcançar seria possível. Obrigado Pai e Mãe.
À minha irmã pelo o apoio que me deu ao longo destes anos e por ser uma inspiração para mim. Desde os tempos de criança sempre foste a pessoa que me quis proteger.
Ao meu sobrinho Martim por ser uma alegria para mim.
Aos meus amigos do grupo Pois é, por estes anos de amizade e por estarem sempre presentes quando precisei.
Aos meus amigos do Memorable Summer por me fazerem rir quando precisei e por estarem sempre presentes nesta fase académica.
Aos meus colegas de estágio, João Palma e Vitor Figueiredo por terem tornado esta experiência muito positiva. Foi um prazer estar ao vosso lado nestes meses.
À Residência Fátima pelos excelentes anos a viver com vocês.
Aos meus orientadores, tanto da faculdade como da empresa, por toda a disponibilidade e paciência que tiveram ao longo deste estágio.

Aos meus Pais, Irmã e Sobrinho

Resumo

Atualmente nas organizações tornou-se essencial adotar *Internet of Things* (IoT) em diversos setores de negócio. Essa adoção nas organizações empresariais foi causada pelos recentes avanços tecnológicos, desde o aumento das capacidades do hardware até ao processamento na nuvem. Este conceito não é apenas utilizado nas máquinas, mas também no auxílio das pessoas que executam processos industriais. Outra componente importante é a forma como conseguimos visualizar a informação gerada pelo o sistema IoT, por isso, AR (AR – *Augmented Reality*) é um complemento na forma como podemos visualizar os dados. AR permite não só reproduzir objetos 3D num mundo real, mas também que haja uma interação entre informação virtual e o mundo real.

O setor empresarial responsável por fazer serviços de manutenção e reparação de equipamentos começou a necessitar cada vez mais de uma interação entre um sistema IoT e AR com objetivo de otimizar custos de manutenção e reduzir o tempo na reparação dos mesmos. Quando um técnico se dirige a um local, grande parte do tempo é gasto no diagnóstico do problema. No entanto, a convergência entre IoT e AR permite que exista uma maior eficiência na reparação de uma máquina.

Este projeto teve como objetivo desenvolver uma solução móvel que permita auxiliar os técnicos na manutenção de um equipamento através de AR e IoT. Quando um técnico se dirige ao local, pode usufruir de AR para visualizar e posicionar o modelo 3D do equipamento, através de um dispositivo móvel. A vertente de IoT veio a adicionar a essa visualização os valores dos sensores que foram armazenados na nuvem depois de terem sido enviados pelos dispositivos.

Numa fase inicial do projeto foi necessário perceber o enquadramento do tema e como as tecnologias de IoT e AR poderiam ser usadas para resolver o problema proposto. Essa investigação serviu também para perceber que requisitos funcionais que o sistema deveria conter.

Na segunda fase foi feito o desenho arquitetural do sistema e da aplicação para definir como o sistema IoT seria constituído, que serviços da nuvem seriam necessários e como o sistema ia comunicar com a aplicação. Para além disso, foi feito o desenvolvimento do projeto, tendo sido implementadas as funcionalidades propostas. Essa fase consistiu no desenvolvimento das componentes do sistema IoT bem como o desenvolvimento da aplicação nativa.

Por último, foram feitos testes à aplicação, também incluindo os de forma a perceber como os utilizadores interagiam com o sistema. Ainda foram feitos testes de carga para perceber como o sistema respondia a um aumento do número de dispositivos.

Palavras-chave: Desenvolvimento móvel, *Internet of Things*¸ Realidade Aumentada, Modelação 3D, Técnico.

Abstract

Currently, in business organizations it has become essential to adopt Internet of Things (IoT) in various business sectors. This adoption has been driven by recent technological advances, from hardware to cloud processing. This concept is not only used on machines, but also to help people performing industrial operations. Another important component is how we get the data generated by the IoT system, so AR (AR - Augmented Reality) is a complement to how we can view information. AR allows not only to reproduce 3D objects in a real world, but also to have an interaction between virtual information and the real world.

The business sector responsible for equipment maintenance and repair services aims to optimize maintenance costs and reduce repair time. When a technician goes to a location, much of the time is spent in diagnosing the problem. However, with the convergence of IoT and AR there is an increase in efficiency in the repair time of a machine.

This project aimed to develop a mobile solution that would help technicians to perform equipment maintenance using AR and IoT. When a technician goes to the location, it can use AR to view the 3D model of the equipment through a mobile device. The IoT component added to the 3D object the values of the sensors that were stored in the cloud after being sent by the devices.

On an early project stage, it was necessary to frame the theme and how IoT and AR technologies could be used to solve the proposed problem. This investigation also served to realize what functional requirements the system should contain.

On a later stage, the architecture design of the system and application was made to define how the IoT system would be created, what cloud services would be needed and how the system would communicate with the application. In addition, a project was developed, and the proposed functionalities were implemented. This stage consisted of the development of the IoT system components as well as of the native application.

In the end, the application was tested, also including tests necessary to understand how users interacted with the system. In addition, load tests were also performed to understand how the system responded in case of an increase in the number of devices.

Keywords: Mobile Development, Internet of Things, Augmented Reality, 3D Modeling, Technician

Conteúdo

Lista de Figuras

Lista de Tabelas

Gráficos

Abreviaturas

ACID – *Atomicity Consistency, Isolation, Durability*

AMQP- *Advanced Message Queuing Protocol*

API - *Application programming interface*

AR - *Augmented Reality*

GPS - Global *Positioning System*

HTTP - *Hypertext Transfer Protocol*

HTTPS - *Hypertext Transfer Protocol Secure*

IaaS - *Infrastructure as a Service*

IoT - *Internet of Things*

IDE - *Integrated development environment*

IIRA - *Industrial Internet Reference Architecture*

JSON - *JavaScript Object Notation*

MQTT - *Message Queuing Telemetry Transport*

MR - *Mixed Reality*

MVC - *Model View Controller*

MVP - *Model View Presenter*

MVVM – *Model View View Model*

PaaS - *Platform as a service*

REST - *Representational State Transfer*

SaaS - *Software as a service*

SLAM - *Simultaneous Localization and Mapping*

SDK - *Software Development Kit*

SOA - *Service-oriented architecture*

SQL - *Structured Query Language*

SSL - *Secure Sockets Layer*

VR - *Virtual Reality*

XML - *Extensible Markup Language*

XMPP - *Extensible Messaging and Presence Protocol*

Capítulo 1

Introdução

Atualmente qualquer atividade de negócio independente dos objetivos tangíveis ou intangíveis, faz uso de tecnologia para encontrar soluções que possam beneficiar o seu negócio a encontrar resultados o mais rapidamente possível. À medida que essa tecnologia avança, diminuem as barreiras entre o mundo virtual e o mundo real.

Internet of Things (IoT) é um conceito que propõe a inclusão de objetos físicos como uma nova forma de comunicação, conectando-os com diversos sistemas de informação. Os casos de uso de IoT geralmente referem-se a objetos com uma certa funcionalidade integrada, seja ela simples ou não, como por exemplo controlar a temperatura do ambiente ou obter valores de uma máquina [6]. IoT ainda é um conceito recente que está a ter um grande impacto na forma como obtemos informações sobre os objetos à nossa volta. Segundo [7], em 2020 estima-se que cerca de 50 milhares de milhões de dispositivos estejam ligados a internet.

A realidade aumentada (AR – *Augmented Reality*) permite visualizar um objeto virtual no mundo real, através de um dispositivo. As empresas estão cada vez mais a adotar essa tecnologia porque permite criar algo único e personalizado de forma a criar mais impacto perante os seus clientes. Além disso, AR e IoT estão em constante em evolução nas diversas áreas de negócio, pois permite otimizar setores desde a indústria até a prestação de serviços.

Com a crescente popularização do smartphone nos últimos anos [8], hoje em dia é possível encontrar muitas aplicações móveis em diversas áreas com a capacidade de fornecer funcionalidades baseadas em realidade aumentada. Contudo, visualizar modelos 3D baseados em AR e obter dados em tempo real provenientes de equipamentos, faz com que haja a necessidade de desenvolver novas soluções capazes de responder de forma eficiente a problemas.

1.1 Motivação

Os impactos de AR e IoT em *Field Force Management* (departamento responsável por gerir os recursos ou os empregados de uma empresa) centram-se essencialmente na otimização de tempo e custos quando existe necessidade de manutenção de um equipamento pois permite a um técnico obter e visualizar informações difíceis de obter manualmente. Segundo [8], estima-se que aproximadamente 50% do tempo usado na manutenção de um equipamento é gasto exclusivamente na localização/identificação do problema. Além disso, o uso de realidade aumentada e IoT em procedimentos de manutenção, consegue reduzir o tempo de um trabalho e os erros que podem ser cometidos. Reparar módulos visualizando um modelo 3D do equipamento ou obter informações de sensores e instruções de reparo, são exemplos onde podemos adotar AR e IoT. Para além destas funcionalidades, a capacidade de ter uma infraestrutura equipada com sensores, permite antecipar e prevenir futuros problemas, reduzindo significativamente o tempo de inatividade de um equipamento.
A principal motivação deste projeto passa pelo desenvolvimento de uma aplicação móveis para fornecer apoio na resolução e prevenção de problemas num equipamento. Deste modo, ao usar IoT conseguimos obter determinadas informações de um equipamento e visualizar as mesmas juntamente com o modelo 3D do equipamento usando AR numa aplicação móvel. Outra motivação passa pelo suporte de instruções AR de modo o tornar mais eficiente o reparo de um equipamento.

1.2 Objetivos do trabalho

Os principais objetivos para este projeto de Engenharia Informática passam pelo desenvolvimento de uma aplicação móvel capaz de fornecer apoio recorrendo a AR e IoT. Pretende-se que esta solução consiga reduzir custos e tempo de manutenção na resolução e prevenção de determinados equipamentos. Posto isso, é proposto o desenvolvimento de uma aplicação móvel com a capacidade de fornecer apoio na resolução e prevenção de problemas num equipamento. O técnico consegue visualizar na aplicação o modelo 3D do equipamento com respetivas instruções e os relatórios anteriormente gerados. Existe ainda duas componentes de inovação que permite o técnico visualizar diferentes perspetivas um modelo 3D e ter a capacidade de prever futuros problemas através de informação obtida pelos sensores IoT embutidos num equipamento.

1.3 Contribuições

O desenvolvimento deste trabalho permitiu que fosse possível integrar um sistema IoT e AR, de forma a possibilitar o uso das duas em simultâneo. Através de um sistema de sensores numa máquina ligados a nuvem e posteriormente uma aplicação que recebe informação e reproduz o equipamento num modelo 3D com os respetivos valores. Os valores gerados nos sensores podem originar anomalias que consequentemente podem causar problemas futuros ou afetar o desempenho da máquina. Assim, um técnico consegue beneficiar de uso de tecnologia capaz de auxiliar nas tarefas de reparação e manutenção de máquinas, conseguindo detetar o principal problema que está a afetar o desempenho ou o próprio funcionamento da máquina. Como consequência secundária da interação de IoT e AR no auxílio dessas tarefas, é possível otimizar e não só o tempo de reparação, mas também otimizar os custos associados.

Como resultado deste trabalho, foi possível ter um capítulo de um livro intitulado "*Industrial and Artificial Internet of Things with Augmented Reality*" aceite para publicação no Handbook Springer-Verlag de título "Convergence of Artificial Intelligence and Internet of Things". Neste capítulo é referida a integração não só de IoT com AR, mas também de AI no auxílio de tarefas em diversos setores da industria, sendo também descritos conceitos como o de *Industrial Internet of Things* e *Industrial Augmented Reality*.

1.4 Estrutura do documento

O documento encontra-se dividido em 3 capítulos. O **Capítulo 1** é referente à apresentação do documento, contendo uma breve descrição do projeto, a sua motivação, instituição de acolhimento e os principais objetivos deste projeto. No **Capítulo 2** são apresentadas diferentes tecnologias e abordagens de desenvolvimento que poderiam ter sido utilizadas no projeto. Neste capítulo também são referidos exemplos de diversas implementações de AR e IoT. No **Capítulo 3** é referido como foi planeado todo o projeto e que metodologias foram adotadas considerando a usada pela Accenture. No **Capítulo 4** é exposto as principais funcionalidades que deveriam ser consideradas no desenvolvimento da aplicação, apresentado assim requisitos funcionais e não funcionais e respetivas descrições de uso. No **Capítulo 5** é apresentado o padrão arquitetural do sistema, decisões técnicas e que tecnologias foram usadas no desenvolvimento. Também foi enunciado que considerações de segurança foram implementadas na aplicação. No **Capítulo 6** é descrito todo o processo relativo à implementação das funcionalidades, bem como a estrutura da aplicação. No **Capítulo 7** foram realizados testes tanto a aplicação como as vertentes de IoT e AR. O **Capítulo 8** é referente ao trabalho que foi realizado ao longo do projeto, os principais obstáculos encontrados e o trabalho futuro.

1.5 Instituição de acolhimento

A Accenture é uma empresa global de consultoria de gestão, tecnologia de informação e outsourcing. Segundo [9], a Accenture está dividida em 5 áreas de negócio: *Strategy, Consulting, Digital, Technology* e *Operations*. Presta serviço a mais de 120 clientes e em mais de 40 setores da indústria como a Banca, Energia, Saúde Transportes, entre outras. O projeto foi desenvolvido nas instalações da Accenture e supervisionado por Sérgio Davide Gaio, responsável pelo departamento de *Field Force Management*.

Capítulo 2 Definição e conceitos

2.1 Realidade Aumentada

É importante perceber a diferenças entre realidade aumentada, realidade virtual e realidade mista (MR – *Mixed Reality*), pois todas têm a capacidade de alterar a nossa perceção sobre o mundo real. A realidade virtual é uma experiência criada por computador que permite um utilizador interagir com um ambiente virtual. A realidade aumentada é a sobreposição de conteúdo digital no mundo real. É a tecnologia mais acessível, pois as pessoas podem usar os seus smartphones ou tablets para executar aplicações de AR [2]. Após perceber estas duas diferenças, a realidade mista diz respeito ao facto de o objeto virtual estar sobreposto no ambiente real permitindo a interação com esse mesmo objeto. Na Figura 2.1, conseguimos visualizar as diferenças entre VR, AR e MR.

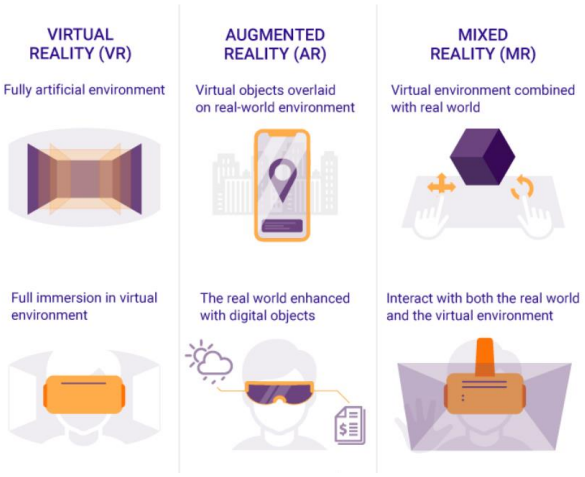

Figura 2.1- VR, AR e MR, retirado de [2]

2.1.1 AR *Tracking*

Baseado em marcadores:
É considerado um dos métodos mais usados para realidade aumentada e é considerada uma das técnicas mais precisa e robusta. Estes tipos de marcadores contém um identificador único codificado e são colocados num objeto ou num determinado local permitindo o reconhecimento por parte de uma aplicação. Ao utilizar a câmera para detetar um marcador primeiramente a imagem é convertida numa escala de cinza para facilitar o processamento da mesma, e seguidamente é calculada a posição da câmera.
De acordo com [10], os marcadores tipicamente usados em AR são quadrados pretos e brancos com figuras geométricas. Isso deve-se ao facto do uso de preto e branco dar um grande contraste em comparação com o ambiente em redor, podendo assim ser reconhecido rapidamente. Atualmente, o SDK Vuforia já permite personalizar um marcador, permitindo atribuir um marcador único personalizável a cada objeto [11].

Baseado em sensores:
Este método é baseado em sensores, e utiliza sensores incorporados nos dispositivos móveis como o GPS, acelerómetro ou giroscópio, para calcular a posição e orientação de uma câmera. Segundo [12], apesar de estes sensores utilizarem um baixo custo energético e computacional, apresentam desvantagens como baixa precisão, limitações do GPS em determinados ambientes, dificuldade em garantir a apresentação dos objetos virtuais e a estabilidade de uma experiência de AR.

Natural Feature Tracking:
Este método permite usar características naturais dos objetos como marcadores. É possível extrair determinados pontos de referência, bordas, cantos ou determinados segmentos, através de imagens que são capturadas pela câmera.
Inicialmente são detetados os "pontos chave" através de um método designado de FAST (*Features from accelerated segment test)*, permitindo detetar e extrair pontos importantes e onde é calculado a orientação desses pontos. Seguidamente é criado um ficheiro com esses pontos e comparado o ficheiro armazenado na base de dados [13]. Após essa comparação é possível contruir um modelo 3D do objeto. Segundo [12], este método ainda está em desenvolvimento pois requer um grande custo computacional e técnicas de visão computacional. Contudo, tem grandes vantagens quando estamos em ambientes desconhecidos ou sem marcadores de referência.

2.1.2 Software Development Kit

Com o aumento de desempenho do hardware nos dispositivos móveis, foram surgindo inúmeros *Software Development kit's* (SDK´s) para AR. De acordo com [10], os SDK's de realidade aumentada permitem introduzir funcionalidades dentro de uma aplicação: reconhecimento, *tracking* e *rendering*. A componente de reconhecimento funciona como o "cérebro" da aplicação AR, a componente de *tracking* os "olhos" e *rendering* diz respeito a objetos imaginários onde é possível visualizar através do dispositivo móvel. Nesta secção é feita uma referência aos principais SDK's.

ArCore:
ArCore é uma *framework open source* da Google para criar aplicações de AR. Segundo [3], o *ArCore* utiliza três recursos para integrar o conteúdo virtual ao mundo real: *motion tracking, environmental understanding, light estimation*.
Motion tracking utiliza a câmera e sensores como o acelerómetro e giroscópio para encontrar a posição e orientação relativa de um utilizador, tornando possível deslocar-se para diferentes locais que o objeto estará na sua posição inicial. O *environmental understanding* permite que o dispositivo móvel detete o tamanho e a localização de todos os tipos de superfícies: horizontais, verticais e angulares. Por último o *light estimation* permite que o dispositivo móvel calcule as condições de luz do ambiente [3].
Como podemos visualizar na Figura 2.2, outra funcionalidade a considerar do *ArCore* é o *Cloud Anchor* [14], pois permite que diversos utilizadores vejam e interajam com o mesmo objeto virtual no mesmo ambiente real [3]. A principal limitação do *ArCore* passa pela necessidade de ter dispositivos móveis com o sistema operativo Android superior à versão 7.0 e iOS 11.0.

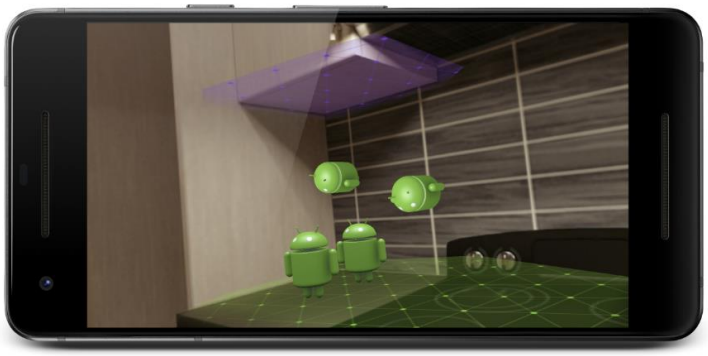

Figura 2.2- Exemplo de ArCore Anchor, retirado de [3]

ArKit:

ArKit é uma framework que combina imagens do muindo real, imagens virtuais e sensores. A primeira versão do ArKit permitia a deteção de imagens 2D. Contudo, com o ArKit 2 foi possível estender essa funcionalidade de modo a permitir tracking de imagens 2D e adicionar a capacidade de detetar objetos 3D comuns, como esculturas, brinquedos ou móveis [15]. O Arkit pode detetar automaticamente superfícies planas através de pontos característicos e correlaciona esses pontos com um vetor. Tanto o *ArCore* como o ArKit têm a capacidade de adicionar um *anchor* permitindo assim que os objetos virtuais permaneçam no lugar onde originalmente foram colocados, independente da posição do orientador. Como podemos observar na tabela 2, o Arkit apenas está disponível para iOS.

Vuforia:

Segundo [16], Vuforia é considerada como um dos SDK's mais utilizados para desenvolver aplicações AR, permite criar e gerir os seus próprios marcadores (VuMark) e tem capacidade de *tracking* de imagens planas, usar a geolocalização e múltiplos alvos simultaneamente. Contêm técnicas de reconhecimento estável de imagens baseadas em visão, permitindo usufruir das capacidades das aplicações móveis. Vuforia permite o *tracking* de 5 objetos simultaneamente, independente das condições de baixa luminosidade. A outra funcionalidade é permitir que a aplicação mantenha uma referência consistente mesmo quando os alvos não estão visíveis.

ArToolkit:

O ArToolKit é considerado como uma das bibliotecas mais utilizadas para o desenvolvimento de aplicações com recursos AR, visto que é uma ferramenta código aberto constantemente atualizada, permitindo assim a exportar para outras plataformas [16]. Inicialmente esta biblioteca foi projetada para ser usada em computadores e não em dispositivos móveis, visto que era muito difícil implementar nesses dispositivos. No entanto, com o tempo foi necessário adaptar essa biblioteca para dispositivos móveis. As funcionalidades passam por ter a capacidade de *tracking* de qualquer padrão que tenha um marcador quadrado, *tracking* suficientemente rápido para usar aplicação de AR em tempo real e suportar múltiplos formatos como RGB, YUV, etc.

Wikitude:
O Witkitude é um SDK que inclui reconhecimento de imagens, *tracking* ou *rendering* de modelos 3D e fornece AR baseada em localização [10]. Segundo [17], o Wikitude utiliza uma abordagem hibrida pois utiliza tecnologias web que permite aos programadores terem experiencias de realidade aumentada em diversas plataformas. Essas experiências utilizam uma interface *ARchictect* para criar objetos em realidade aumentada, podendo assim integrar o SDK numa aplicação móvel através de uma componente especifica designada de *ArchitectView*.
O Wikitude permite carregar objetos 3D, contudo, nenhum dos formatos comuns de arquivos 3D é suportado, pois apenas aceita *Wikitude 3D Format* (. wt3). Para facilitar a conversão de formatos convencionais em .wt3, o Wikitude desenvolveu uma aplicação para computador (Wikitude 3D encoder), permitindo assim importar formatos como *.fbx para *.wt3 [15].
Os recursos recentemente implementados passam pelo *tracking* de um modelo 3D e a funcionalidade de localização e mapeamento simultâneo (SLAM - *Simultaneous Localization and Mapping*).

EasyAr:
EasyAr é uma *framework*, que oferece uma versão gratuita com capacidade de reconhecer e fazer *tracking* de *QR code* e de imagens planas. No entanto, se existir a necessidade de utilizar a funcionalidade de SLAM ou 3D *object tracking* será necessário adquirir a versão paga, no que poderá ser uma limitação desta *framework* [18].

2.2 Internet of Things

De acordo com [6], *internet of Things* (IoT) está cada vez mais a ter impactos nas nossas vidas pois todos os aparelhos ou dispositivos já estão ligado em rede de modo a comunicar entre si. Existem setores como assistência médica, segurança, vigilância e gestão de produtos que utilizam tecnologias baseadas em IoT, permitindo que exista uma grande quantidade de novos serviços, dos quais responderão às necessidades dos utilizadores [19] .
Segundo [19], devemos considerar oito características que um sistema IoT deve conter:

1. Heterogeneidade: Devido ao facto de existirem diversos dispositivos e protocolos.
2. Escalabilidade: Devido ao número massivo de serviços que devem estar disponíveis
3. Troca de dados ubíqua: Permite a troca de informação através de tecnologias Wireless. Este é um ponto importante na disponibilidade de um serviço, pois pode trazer problemas quando não existe comunicação entre os dispositivos e os serviços.
4. Otimização de soluções de energia: Como existe uma enorme variedade de dispositivos e comunicações entre eles, minimizar a energia a ser usar nessas comunicações é uma limitação primária.
5. Capacidade de localização e *tracking*: as entidades de IoT podem ser identificadas e fornecer comunicações sem fios de curto alcance através da localização dos dispositivos.
6. Capacidade de auto-organização: Os nós de uma rede de IoT tem a capacidade de se organizar autonomamente em redes *ad hoc* de modo a fornecer os meios necessários para executar as tarefas e partilhar dados sem intervenção de um utilizador.
7. Interoperabilidade semântica e gestão de dados: Devido à troca massiva de dados e da diversidade de dispositivos, existe a necessidade de garantir a interoperabilidade entre diferentes aplicações, fornecendo formatos, linguagens e padrões adequados.
8. Segurança e privacidade: Um sistema IoT deve garantir segurança e preservar a privacidade dos dados.

2.2.1 Arquitetura

A medida que as aplicações IoT foram crescendo em implementações de software, houve necessidade de criar arquiteturas de referência para a indústria. O principal objetivo era facilitar a interoperabilidade e simplificar o desenvolvimento. No entanto, não existe um consenso sobre uma arquitetura para IoT que seja aceite por todos. Segundo [4], podemos classificar as arquiteturas com base em camadas, sistemas e interações.

Três a cinco camadas:

De acordo com [4], inicialmente surgiu a arquitetura de três camadas, considerada como a mais simples, porque representa a ideia básica de IoT sendo composta pelas camadas: perceção, network e aplicação. A primeira refere-se à camada física, onde estão situados os sensores para detetar e obter informações. A segunda camada é responsável por ligar os dispositivos da rede e os respetivos servidores, podendo ser usada para transmitir dados. Por fim a camada de aplicação fornece os serviços específicos ao utilizador. Devido ao facto desta camada não ser suficiente em muitos modelos de IoT, foi necessário criar uma arquitetura desta vez com cinco camadas mantendo as de perceção e aplicação, e acrescentando as camadas de transporte, processamento e negócio, como podemos observar na Figura 2.3. A camada de transporte é responsável por transmitir os dados recebidos da camada de perceção para a camada de processamento. São utilizadas diversas formas de transmissão como *wireless* ou bluetooth, utilizando protocolos como por exemplo o IPv6 [20]. A camada de processamento armazena, analisa e processa principalmente as informações provenientes da camada de transporte. Pode ser designada como um middleware pois consegue gerir e fornecer serviços para camadas inferiores. Nesta camada são utilizadas tecnologias como computação na nuvem, base de dados ou processamento de grandes quantidades de dados [4].

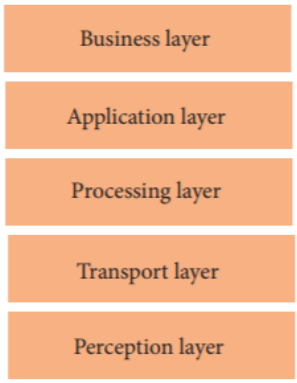

Figura 2.3- Arquitetura de cinco camadas, retirado de [4]

Arquitetura baseada em Nuvem e Fog:

De acordo com [21], o modelo de computação na nuvem é uma alternativa muito eficiente ao centro de processamento de dados para clientes que utilizam aplicações na *web* ou grande quantidade de dados. Em algumas arquiteturas de sistema, o processamento dos dados é feito através de computadores na nuvem pois assim se consegue obter flexibilidade e escalabilidade. Computação na nuvem tipicamente está dividido em três níveis: *Software as a Service* (SaaS), *Platform as a Service*(PaaS), *Infrastructure as a Service*(IaaS) [21]. Neste sistema a nuvem armazena e analisa todos os dados que recebe, no entanto, de modo a evitar isso foi projetado a computação *fog*.

Fog computing é uma plataforma virtualizada que fornece serviços de computação, armazenamento entre dipositivos finais utilizando computação na nuvem [22]. Segundo [4], este novo modelo apresenta uma abordagem em camadas onde os sensores e as *gateways* que fazem parte do sistema contendo a capacidade de processar e analisar os dados. Como podemos observar na Figura 2.4, esta arquitetura contém cinco camadas designadas de transporte, segurança, armazenamento, pré-processamento, monitorização e física. A camada de monitorização, controla os recursos, a energia e as respostas ao serviço [4]. A camada de pré-processamento funciona como um filtro dos dados nas extremidades de um sistema.

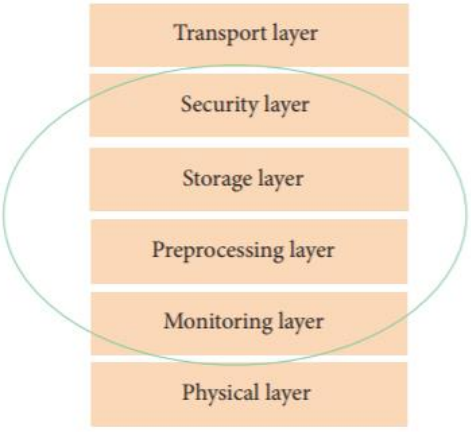

Figura 2.4- Arquitetura fog, retirado de [4]

Social IoT:
Segundo [4], temos de considerar ligações entre os diversos dispositivos da mesma forma que os humanos formam relações. Existem três princípios do sistema *Social Internet of Things* (SIoT). O primeiro princípio assenta na possibilidade de navegar pela rede, de modo a descobrir os dispositivos e serviços de forma eficaz e garantindo a escalabilidade. O segundo princípio dá enfase ao grau de fiabilidade para tirar vantagens das interações entre dispositivos "amigos". No último princípio os modelos projetados para estudar as redes sociais podem ser reutilizados para tratar de questões relacionadas à IoT (intrinsecamente relacionadas a redes extensas de objetos interconectados) [23].
Tendo em conta estes princípios foi proposta uma arquitetura baseada nas relações sociais. Segundo [4] a arquitetura do lado de servidor possui três camadas, sendo a primeira a conter uma base de dados que armazena os detalhes dos dispositivos, como os seus atributos, meta informação, e os seus relacionamentos. A segunda camada contém o código para interagir com os dispositivos, consultar o seu estado e usar um subconjunto deles para fornecer um serviço. A terceira camada é a da aplicação, que fornece serviços aos utilizadores. Do lado do dispositivo, temos duas camadas. A primeira permite que um dispositivo se ligue a outro dispositivo e troque informações. A outra camada designada de camada social permite executar pedidos e interagir com a camada de aplicação do servidor.

2.2.2 Sensores e atuadores

Muitas atividades industriais, contêm sensores responsáveis por detetar mudanças num ambiente/equipamento, normalmente são usados em fábricas, em casas de clientes, em linhas de montagens ou até mesmo em locais remotos. Por exemplo em áreas como a de prestação de serviços de manutenção ou reparação de equipamentos a capacidade usar sensores em diversas máquinas permite perceber quando existe problemas, ou até mesmo antecipá-los. Muito dos sensores usados em IoT

normalmente são pequenos, tem baixo custo e consomem pouca energia. Primeiramente vamos falar de sensores que utilizamos diariamente que estão embutidos no nosso telemóvel. Um smartphone é um dispositivo útil e de fácil utilização que possui uma série de recursos integrados de comunicação e processamento de dados [4]. Tipicamente esses dispositivos tem sensores como acelerómetro, que permite medir a velocidade em três dimensões, o giroscópio que deteta a orientação de um utilizador, câmera, magnetómetro que deteta campos magnéticos, GPS, sensor de luz, e os mais recentes já possuem sensores de humidade. Contudo, dependendo da área de negócio, o propósito de um sensor é diferente. Por exemplo, pode ser usado um sensor de pressão (barómetro e manómetro) num ambiente industrial para detetar e medir a pressão de um gás, ou um sensor ótico permitindo obter informações sobre fibra ótica.
Quando falamos de sensores, temos de referenciar possíveis atuadores. Um atuador é um dispositivo que pode afetar uma mudança num ambiente, convertendo energia elétrica em algum tipo de energia útil [4]. Alguns exemplos de atuadores são equipamentos de arrefecimento, altifalantes, luzes e monitores. Um atuador pode também alertar um utilizador sobre algum problema através de um alarme ou notificação.

2.2.3 Protocolos da camada de aplicação

A camada de aplicação funciona como uma abstração que engloba protocolos que permitem comunicações entre diversas aplicações. Os protocolos tradicionais usados na camada de aplicação não eram opção para serviços de IoT devidos as restrições das redes de baixa potência com perdas (LLNs - *Low power and Lossy Networks*) [24]. Contudo, para solucionar esse problema, segundo [5], foram criados seis protocolos na camada de aplicação utilizados para atualizar servidores com valores atuais dos dispositivos finais e transportar instruções das aplicações para esses dispositivos.

Constrained Application Protocol (COAP):
CoAP é um protocolo de pedido/resposta projetado para aplicações *Machine-to-Machine* (M2M) de forma a garantir a interoperabilidade usa *Hypertext Transfer Protocol* (HHTP) contendo comandos como do HTTP.
Para remover a sobrecarga de protocolos de comunicação como o *Transmission Control Protocol* (TCP) e diminuir requisitos da largura de banda é utilizado o protocolo UDP [5]. Como o CoAP é construído sobre o protocolo *User Datagram Protocol (UDP)* [25] e não TCP, não permite utilizar *Secure Sockets Layer* (SSL) e *Transpor Layer Security* (TLS) [26] para comunicações seguras, contudo, utiliza o *Datagram Transport Layer Security* para fornecer as mesmas garantias de segurança [5].

Message Queue Telemetry Transport (MQTT):
MQTT é um protocolo de mensagens baseado em publicador/subscritor projetado para comunicações M2M. Cada dispositivo pode ser um publicador que envia mensagens baseadas em tópicos para um *broker* ou barramento de eventos e um subscritor recebe essa mensagem automática sempre que houver uma atualização num tópico que esteja subscrito [5]. Como exemplo, podemos visualizar na Figura 2.5 onde o cliente A publicou o tópico temperatura no broker e o cliente C e B subscreveram esse tópico.
Embora o MQTT seja executado por TCP, ele foi projetado para ter baixo *overhead* comparado com outras aplicações TCP, tornando-se assim um protocolo adequado para IoT comparativamente ao método de pedido/resposta do CoAP. Como o CoAP é baseado em UDP tem menos *overhead*, contudo a probabilidade de uma perda de pacote é maior comparativamente ao TCP [5].
Outro ponto importante, é a fiabilidade do MQTT pois garante que uma mensagem seja entregue ao destinatário. No MQTT existem 3 níveis de qualidade de serviço (QoS): *fire and forget, delivered at least once, delivered exactly once* [5].

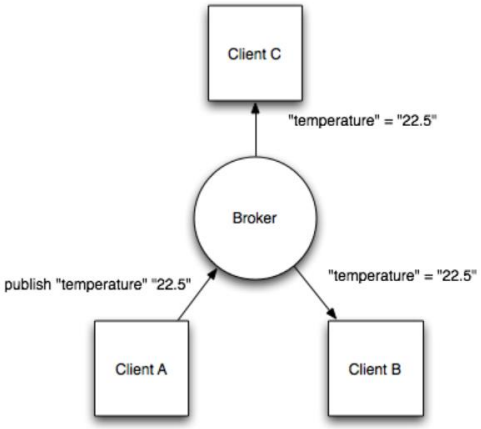

Figura 2.5- Modelo publicador/subscritor, retirado de [5]

The Extensible Messaging and Presence Protocol (XMPP):
Segundo [5], o XMPP é utilizado há mais de uma década e tem sido utilizado por toda a internet, e devido ao facto de ser um protocolo antigo, poderá ser insuficiente para fornecer serviços necessários a novas aplicações. Contudo, tem vindo a recuperar muito atenção como protocolo utilizado para IoT. O XMPP utiliza o protocolo TCP suportando a arquitetura publicador/subscritor e também pedido/resposta. Permite comunicações em tempo real quendo se trata de pequeno volume de mensagens, garantido segurança através de TLS/SSL, no entanto, como podemos visualizar na figura 10 este protocolo não fornece opções de QoS o que é pode tornar impraticável para comunicações M2M.

Restful Services:
O *Representational State Transfer* (REST) [5] não é um protocolo, mas sim um estilo arquitetural que usa métodos HTTP GET, POST, PUT, e DELETE para fornecer um sistema de mensagens pedido/resposta. O REST é importante em IoT pois é suportado por todas as plataformas de nuvem M2M (Amazon, Google, etc), e pode ser facilmente implementado em dispositivos móveis [5]. Contudo, a maioria das plataformas M2M não suportam HTTPS (*Hyper Text Transfer Protocol Secure*), em vez disso, fornecem chaves de autenticação. Outra limitação para que esta arquitetura se torne um protocolo muito pouco usado é devido ao facto de ser difícil de implementar.

Advanced Message Queuing Protocol (AMQP):
O AMQP surgiu no setor financeiro, pois é um padrão utilizado para passar mensagens entre aplicações e organizações. Ele tem a capacidade de ligar sistemas, processos de negócio de transmitir de forma confiável as instruções pedidas [27]. Segundo [5], este protocolo pode utilizar diversos protocolos de transporte, mas preferencialmente usa TCP para assegurar fiabilidade. Utiliza comunicação publicador/subscritor de mensagens. Quando existem interrupções na rede este protocolo garante fiabilidade ao armazenar e encaminhar as mensagens. Essa fiabilidade é garantida através de três abordagens de entrega de mensagens [5]: *at most once, at least once, exactly* [5].
A limitação deste protocolo está relacionada ao baixo sucesso quando existe uma baixa largura de banda.

Websocket:
O protocolo *Websocket* [5] foi criado com o intuito de reduzir o overhead de comunicação da internet, fornecendo comunicação *full-duplex* (transmissão de dados simultaneamente em ambos os sentidos) em

tempo real [5]. O *Websocket* não é um protocolo baseado em pedido/resposta nem em publicador/subscritor, sendo executado sobre o TCP permitindo usar TLS/SSL para fazer comunicações. Contudo, para a vertente de IoT existe um subprotocolo nomeado de *Websocket Application Messaging Protocol* (WAMP) [28], que permite adaptar o padrão publicador/subscritor [5]. Este protocolo tem a mesma limitação que o XMPP e REST, pois ambos não fornecem QoS.

Protocol	Transport	QoS options	Architecture	Security
CoAP	UDP	YES	Request/Response	DTLS
MQTT	TCP	YES	Publish/Subscribe	TLS/SSL
XMPP	TCP	NO	Request/Response Publish/Subscribe	TLS/SSL
REST	HTTP	NO	Request/Response	HTTPS
AMQP	TCP	YES	Publish/Subscribe	TLS/SSL
Web socket	TCP	NO	Client/Server Publish/Subscribe	TLS/SSL

Figura 2.6- Protocolos de comunicação, retirado de [5]

Na Figura 2.6, é possível ver a comparação entre as características dos diferentes protocolos.

2.2.4 Middleware

Computação ubíqua é o núcleo de modelo de IoT, permitindo assim que todos os dispositivos estejam ligados entre si. Para garantir isso tem de existir interoperabilidade e padrões bem definidos nesses dispositivos heterogéneos. A solução para abstrair esses padrões e formatos é ter uma plataforma de *middleware*, que funciona como uma ponte de software entre os dispositivos e a aplicação. Segundo [4], existem diversas abordagens de *middleware* e podem ser classificados consoante o seu desenho:

Baseado em eventos: É uma arquitetura baseada em eventos, composta por dois componentes, clientes e *brokers* de eventos. Os clientes podem ser publicadores de eventos ou subscritores de eventos e utilizam os serviços fornecidos pelo middleware para comunicar. Os *brokers* representam o middleware e fornecem uma implementação distribuída [29].

Orientado a serviços: É baseado na arquitetura orientada a serviços (SOA - *Service Oriented Archictures)* , contendo módulos independentes que fornecem serviços através das interfaces disponíveis. Determinados recursos são abstraídos por um conjunto de serviços usados pela aplicação. Existe um repositório, onde os serviços são colocados e os "consumidores" conseguem aceder a esses serviços.

Orientado a base de dados: Nesta abordagem, a rede de dispositivos IoT é considerada como um sistema de base de dados. Através de uma interface, as aplicações podem consultar a base de dados usando linguagem de consulta.

Semântica: É baseado na interoperabilidade entre diferentes dispositivos. Os dispositivos incorporam diferentes formatos de dados e ontologias e estão ligados a uma *framework,* permitindo a troca desses dados. Para cada dispositivo é necessário um adaptador para mapear N padrões para um padrão abstrato. Essa técnica permite que vários recursos físicos comuniquem entre si, mesmo que os dispositivos não implementem os mesmos protocolos [4].

Aplicação especifica: Este tipo de middleware é usado especificamente no domínio da aplicação para a qual é desenvolvido, permitindo assim ajustar a arquitetura com base nos requisitos.

2.3 Abordagens de desenvolvimento móvel

O desenvolvimento de aplicações móveis é um processo complexo e que exige conhecer as diferentes abordagens existentes. Segundo [30], ao desenvolver aplicações móveis existem certos fatores que devem ser tidos em conta, como a integração com o hardware do dispositivo, a segurança, o desempenho, a confiabilidade e as limitações de armazenamento. Cada fator pode ser decisivo para o bom funcionamento e sucesso de uma aplicação pois cada uma oferece as suas próprias vantagens. Nesta secção vamos referir duas abordagens de desenvolvimento: nativo e de multi-plataforma.

Desenvolvimento nativo:
As aplicações nativas são criadas usando SDK's e ferramentas especificas de desenvolvimento para cada plataforma, como por exemplo, java para android e *objective C* para iOS. De acordo com [31], as vantagens das aplicações móveis nativas é de terem o acesso máximo aos recursos do dispositivo, API's disponíveis em cada plataforma e o melhor desempenho possível. Quando estamos a criar aplicações muito complexas ou que utilizem modelação 3D o desenvolvimento nativo é a melhor abordagem pois requer acesso às *frameworks* e bibliotecas externas [32]. Como declarou a *MGI Research* intitulado de Guia do comprador para Plataformas Móveis de Aplicações Empresariais (MEAP), a arquitetura nativa tende a oferecer uma experiencia de utilizador mais "rica", mais envolvente graficamente, alto desempenho e com a capacidade de integrar com funções e *back-end* de sistemas empresariais [31]. As principais desvantagens referem-se: (i) ao tempo de desenvolvimento de uma aplicação, pois é necessário desenvolver para as diversas plataformas, (ii) aos custos e (iii) às diversas linguagens de programação que são necessárias adquirir.

Desenvolvimento multi-plataforma:
De acordo com [33], existem quatro abordagens multi-plataforma para aplicações móveis: Híbrida, *Web*, *Interpreted* e *Cross Compiled.*

Abordagem Híbrida:
A abordagem híbrida é considerada como web e nativa pois recorre a tecnologias como HTML, CSS e JavaScript, e é executada dentro de uma máquina virtual de um dispositivo móvel. De acordo com [33], as principais vantagens desta abordagem são poder reutilizar a interface do utilizador em diferentes plataformas nativas, pois a lógica de negócio é independente da plataforma, e ter a capacidade de usar os recursos do dispositivo. As principais desvantagens passam pelo desempenho quando comparado a aplicações nativas, pois necessitam de um navegador *web* para executar. Como podemos visualizar na Figura 2.7, a abordagem hibrida faz uso de uma camada de abstração de JavaScript e com isso está mais sujeita a vulnerabilidades embora seja possível reutilizar a interface do utilizador para diversas plataformas, fazendo com que a experiencia do utilizador não seja a melhor.
O *PhoneGap* é um exemplo de uma *framework* que segue uma abordagem híbrida pois é uma ferramenta de desenvolvimento de aplicações móveis, que usa linguagens de programação *web*, como HTML5, CSS3 e JavaScript.

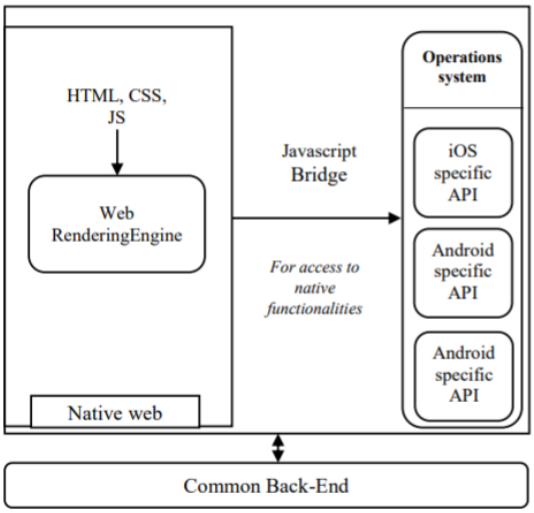

Figura 2.7- Abordagem Híbrida, retirado de [1]

Abordagem Web:

A principal característica deste tipo de abordagem é devido ao facto de uma aplicação móvel não necessitar de ser instalada num dispositivo móvel, pois a sua execução é feita através de um *web browser*.

Segundo [31], existem duas formas de segmentar esta abordagem para dispositivos móveis: *web design responsive* e *mobile web app*. A primeira refere-se ao facto de os programadores focarem-se em apenas modificar o *layout* para este se adaptar a um ecrã de menor tamanho. A única vantagem é ter apenas um único código através de linguagens como HTML, JavaScript e CSS. Contudo, limita a capacidade de criar uma experiência para um dispositivo móvel. Na segunda abordagem os dispositivos móveis são detetados e direcionados para uma aplicação da *web* otimizada, onde é criada uma interface consoante as convenções especificas do dispositivo móvel. A vantagem desta abordagem passa por não ser necessário criar diversas implementações de interfaces com o utilizador.

Para além das vantagens acima referidas, é importante referir que a aplicação e os seus dados estão armazenados no servidor, não sendo necessário qualquer tipo de manutenção no dispositivo móvel. No entanto, isso pode ser uma desvantagem, pois pode ter um desempenho inferior as outras abordagens por estar dependente da rede móvel [33]. Como podemos visualizar na Figura 2.8, no lado esquerdo está representado o navegador do dispositivo móvel, responsável por fazer pedidos ao servidor. No lado direito é onde está armazenado toda a lógica de negócio da aplicação e os respetivos dados.

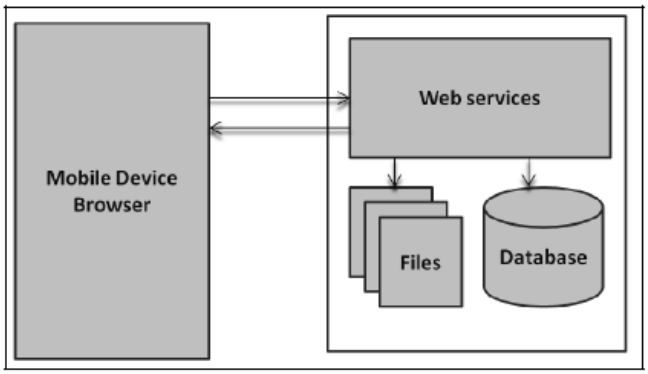

Figura 2.8- Aplicação Web, retirado de [33]

Abordagem *Interpreted:*

Neste tipo de abordagem o código nativo é gerado automaticamente em tempo de execução tendo um interpretador responsável por essa tarefa. O interpretador posteriormente interage com uma camada de abstração para ter acesso as *API's* nativas de cada plataforma (Figura 2.9).

As principais vantagens dessa abordagem são a eficiência ao acesso as interfaces nativas e a reutilização da lógica de negócio para diversas plataformas [34]. Por vezes o desempenho pode ser afetado pelo acesso a essa camada de abstração em tempo de execução.

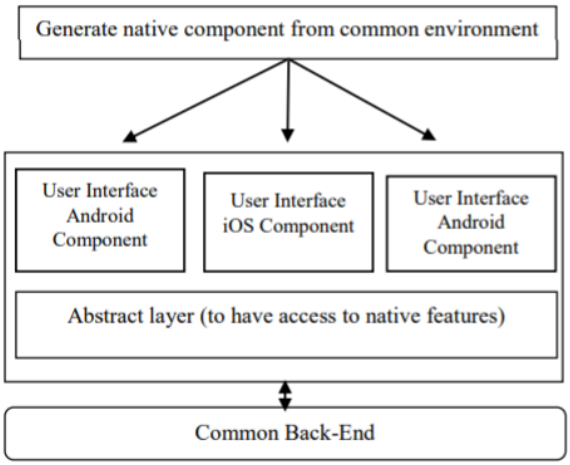

Figura 2.9- Abordagem *Interpreted*, retirado de [1]

Abordagem *Cross Compiled:*

Esta metodologia é multi-plataforma, pois utiliza uma linguagem de programação comum. Como podemos observar na Figura 2.10, esta abordagem faz uso de um *cross compiler* responsável por compilar essa mesma linguagem num código nativo para uma plataforma especifica [34]. A principal vantagem dessa abordagem passa por obter o mesmo desempenho que as aplicações nativas e fornecer todas as características das interfaces nativas. Contudo, esta abordagem não permite o uso de alguns recursos do dispositivo, como a por exemplo a câmera e GPS. Isso acontece devido ao facto dessas funcionalidades serem especificas para cada plataforma [1]. Este tipo de abordagem é mais utilizado para aplicações simples.

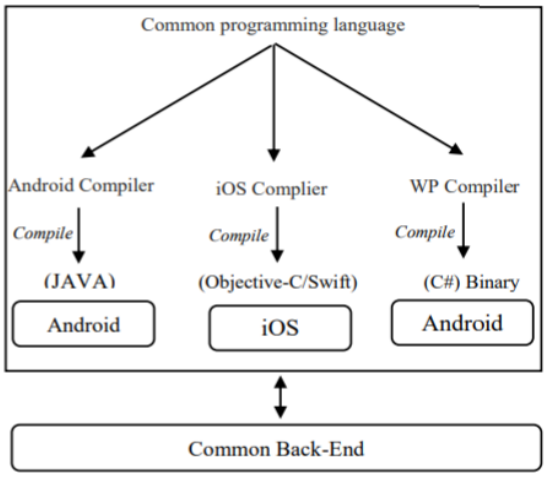

Figura 2.10- Abordagem *Cross-compiled*, retirado de [1]

2.4 Padrões arquiteturais

Em engenharia de software, um padrão arquitetural é uma solução reutilizável para um determinado problema que ocorre num determinado contexto. Por isso, é determinante entender como as partes principais dos sistemas são integrados e como os dados fluem. É importante que exista uma boa estruturação quando se desenvolve um software, e para que tal aconteça, são apresentados de seguida 3 padrões: MVC, MVP e MVVM.

Padrão *Model-View-Controller* (MVC):
Num software onde a interface tende a sofrer mais mudanças que do que a lógica de negócio, foi necessário separar as funcionalidades da aplicação com a interface. Segundo [35], este padrão foi dos primeiros a ser concebidos e é composto por três componentes independentes: *View*, *Controller*, *Model*.
O *Model* é responsável por gerir a lógica de negócio e manipular os dados. Segundo [36], responde a pedidos de informações sobre o seu estado e responde a instruções para alterar esse mesmo estado.
O *Controller* interpreta as ações que o utilizador fornece, informando a camada *Model* e *View* para alterar os dados/estado conforme o solicitado. Por exemplo, pode existir vários *Controllers* cada um com uma funcionalidade diferente liados a um *Controller* principal.
O *View* tem como principal objetivo apresentar os dados ao utilizador, que são manipulados pelo *Controler* e posteriormente pelo *Model*.

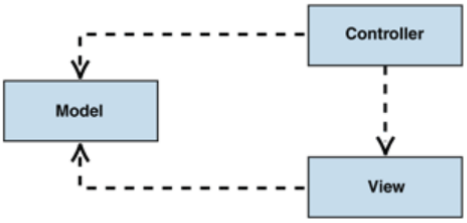

Figura 2.11- Estrutura do MVC, retirado de [36]

A Figura 2.11 representa uma estrutura do MVC, onde podemos observar que tanto o *Controller* como a *View* dependem do *Model*, *o Controller* para atualizar os dados e a *View* para obter esses dados. No entanto o *Model* não depende de ninguém, e este é um dos principais benefícios desta independencia, isto porque permite que o *Model* seja testado independemente sem que afete outras camadas [36].

Padrão *Model-View-Presenter* (MVP)*:*

Segundo [35], o padrão MVP é uma variante do MVC criado especificamente para automação de testes, onde o objetivo seria aumentar a quantidade de código a ser testado. A arquitetura do MVP é descrita em 3 componentes [37]: *Model*, *View* e *Presenter*.
O *Model* tem o mesmo conceito que no MVC, é responsável por responder a pedidos de informações sobre o seu estado e responde a instruções para alterar esse mesmo estado.
A *View* é a componente responsável pela exibição de informações, em Android representa por exemplo *activity.xml*
O que difere do MVC é o a camada *Presenter*. Essa camada tem como objetivo fazer a ligação entre a *View* e o *Model*, respondendo a eventos e alterando o *Model*. No MVC, o controlador é responsável por determinar quais são as *Views* que irão dar resposta a qualquer ação. No MVP, isso difere, porque as ações são encaminhadas pela *View* para o *Presenter*.

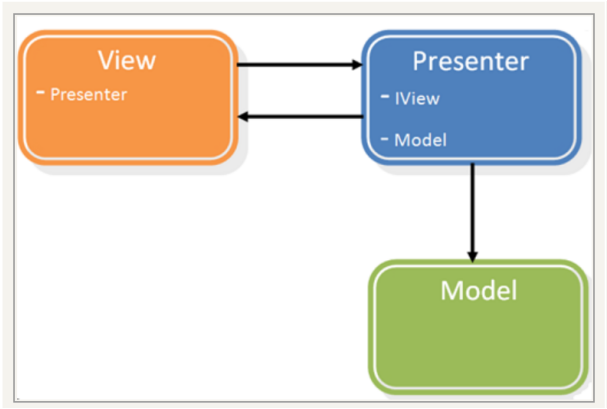

Figura 2.12- Estrutura MVP, retirado de [35]

Como podemos observar, na Figura *2.12* está representada a estrutura MVP. A *View* captura os eventos do utilizador e envia para o *Presenter*. O *Presenter* toma uma decisão e atualiza a *View*. Se for necessário também pode comunica com o *Model* caso seja necessário atualizar informação.
Padrão *Model View View Model* (MVVM):
Segundo [35], o MVVM é um padrão deriva do padrão MVC, contudo inicialmente foi desenhado especialmente para *Windows Presentation Foundation* (WPF). O MVVM é uma arquitetura que é facilmente adaptada para plataformas de desenvolvimento, com o objetivo de separar a interface da lógica da aplicação, onde a *View* é responsabilidade de um designer em vez de um programador. A arquitetura MVVM é composta por três componentes, a *View, View Model* e o *Model*.
O *Model* tem o mesmo conceito que no MVC, é responsável por responder a pedidos de informação.
A *View* é responsável pela exibição da informação no caso de WPF é definida recorrendo ao XAML (eXtensible Application Markup Language). Nesta arquitetura, as *Views* têm uma relação *many-to-one* com os *ViewModels*, ou seja, diferentes *Views* podem utilizar o mesmo *ViewModel*.
A *ViewModel* é um intermediário entre o *Model* e a *View* responsável por gerir a lógica da *View*. Esta componente interage com o *Model* para obter informação e posteriormente envia para à *View* onde irá ser apresentado ao utilizador.

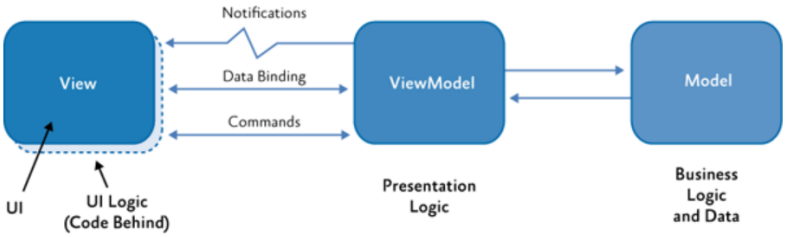

Figura 2.13- Estrutura MVVM, retirado de [38]

O Data Binding representado na *Figura 2.13* é um mecanismo que permite fazer a ligação entre o que é apresentado na *View* e o conteúdo do *Model*. No databinding , o *ViewModel* não precisa de notificar as alterações através de código à *View*, a própria *View* sabe quando um dado é carregado [37] . Em [39], é possível ver um exemplo de uma implementação com base no *Data Binding*. O *ViewModel* implementa propriedades (*PropertyChanged*, ver referência [39]) e comandos aos quais a *View* pode vincular e caso haja alterações notifica essa mesma *View* através de eventos.

2.5 REST e SOAP

Os serviços web estão cada vez mais a ser utilizados como uma nova tecnologia emergente para a comunicação entre aplicações móveis. Os dispositivos móveis podem assim utilizar serviços web para estabelecer comunicação entre um cliente e servidor. A necessidade mais importante hoje em dia é que esses serviços sejam ininterruptos e leves em comunicações móveis [40]. Os serviços web são baseados na arquitetura REST e no protocolo SOAP.

SOAP:

O Simple Object Access Protocol (SOAP) é um protocolo de comunicação utilizado em web services, que fornece um mecanismo simples para troca de informações estruturadas em um ambiente distribuído e descentralizado usando XML [41]. Um dos objetivos do design do SOAP á a sua simplicidade e extensibilidade. Este protocolo pode ser usado em diversos sistemas que vão desde troca de mensagens a RPC (Remote Procedure Calls). RPC define uma convenção que pode ser usado para representar chamadas de procedimentos e respostas.

As mensagens SOAP são compostas por um envelope, cabeçalho e o corpo da mensagem. O envelope identifica o documento XML como sendo uma mensagem SOAP. O cabeçalho contém os atributos opcionais da mensagem e o corpo da mensagem contém informações do pedido e da resposta.

REST:

Representational State Transfer (REST) é uma arquitetura para fornecer padrões entre sistemas de computação na web, de modo a facilitar a sua comunicação. Os serviços web baseados em REST são fáceis de consumir por plataformas móveis, isto porque a comunicação entre cliente e o servidor é realizada através de um único protocolo de pedido e resposta.

Segundo [40], as principais características de uma arquitetura REST em aplicações móveis passam por estas serem stateless (o servidor não armazena nenhum estado sobre a sessão do cliente), minimizando a volatilidade das conexões. É baseado um URI's (Uniform Resource Identifiers), portanto é fácil invocar os serviços. Gerar um pedido com base em REST não implica estar dependente de uma tecnologia, mas para serviços web é comum usar pedidas HTTP através de operações HEAD,POST,PUT,GET e DELETE. As respostas REST, também são baseadas em HTTP permitindo

assim obter um resultado através de códigos específicos e a sua representação pode ser em vários formatos (XML, HTML, JSON). Segundo , REST suporta todo o tipo de dados, não sendo necessário interpretar/formatar as mensagens da mesma forma que o protocolo SOAP, onde é necessário fazer parsing da mensagem XML. Como consequência, serviços da Web REST fornecem uma maior flexibilidade em relação ao tipo de dados retornados.
Apesar de a estrutura SOAP ser mais segura comparativamente a REST, fatores como complexidade de integração relacionada com questões de processamento e níveis de consumo de recursos que se aplicam a aplicações móveis, concluímos que REST demonstra ser a abordagem mais indicada para desenvolver aplicações móveis [42].

2.6 Plataformas de desenvolvimento móvel

Android Studio:
O Android Studio é um *Integrated Development Environment* (IDE) usado para desenvolver aplicações em Java para Android. Este IDE permite ter um sistema de compilação flexível, um emulador, ferramentas de teste, frameworks e ferramentas para analisar o código para a deteção de erros. Por defeito existem três módulos para fornecer o acesso a ficheiros de um projeto: manifest, java, res. O modulo manifest contém todas as permissões e informações importantes, o módulo java contém todos os arquivos de código Java incluído o código de teste e o módulo res guarda todos os recursos que não são código, como imagens, *layout* XML (*Extensible Markup Language*), etc [43].

Xcode:
O Xcode é um ambiente de desenvolvimento para desenvolver aplicações para MacOS. O Xcode fornece ferramentas para que permitem criar uma aplicação até a sua execução fornecendo suporte a linguagens como Swift e C. Integrado a este ambiente de desenvolvimento, podemos visualizar a interface de desenho ao lado do código de implementação, permitindo ter uma noção visual da aplicação [44].

ViroReact:
O ViroReack é uma multi-plataforma pois permite aos programadores desenvolver rapidamente aplicações de realidade aumentada usando o React Native (permite criar aplicações móveis usando JavaScript). Esta plataforma permite reutilizar código para Android e iOS e obtém o máximo de desempenho do hardware para aplicações de realidade aumentada. A plataforma é composta por duas componentes: Mecanismo de *rendering* 3D nativo e uma extensão do *React* para o desenvolvimento de AR e realidade virtual (VR - Virtual Reality). Para o desenvolvimento de AR permite integrar Frameworks como *ArCore* e ArKit [45].

2.7 Base de dados relacionais e não relacionais

Segundo [46], os sistemas tradicionais de base de dados baseam-se no modelo relacional e são conhecidos como *Structured Query Language* (SQL). Contudo ao longo destes anos tem havido um crescente interesse sobre base de dados não relacionais (NoSql), pois a maioria deles são utilizados em *BigData* (grande conjunto de dados gerados e armazenados).

SQL:

As bases de dados tradicionais como SQL, são focadas na consistência e seguem as propriedades ACID (*Atomicity, Consistency, Isolation, Durability*) quando realizam transações. De acordo com [46], atomicidade refere-se a transações que são realizadas ou descartadas, não podendo haver transações parcialmente realizadas. Para existir consistência quando uma transação falha o sistema reverte até ao estado anterior propagando essa informação por todos os nós. No Isolamento as transações são realizadas independentemente, e por último, a durabilidade refere-se às transações poderem ser guardadas em registos de modo a não serem perdidas, caso haja uma necessidade de recuperar o sistema. As bases de dados relacionais contêm um conjunto de tabelas, em que cada uma contém colunas com atributos e cada linha da tabela representa uma instância única de dados. De acordo com [47], para dados estruturados este tipo de bases de dados se enquadram melhor, contudo, existem limitações relativamente à escalabilidade e complexidade. Com o aumento de utilizadores e dados armazenados é necessário que haja forma de garantir escalabilidade do sistema, sendo possível distribuir a base de dados por diversos servidores. No entanto, isso gera problemas, pois este não está preparado para particionar este tipo de dados. Outra limitação é a complexidade de trabalhar com base de dados relacionais, pois requer converter todos os dados em tabelas mesmo quando os dados não são projetados para isso.

NoSQL:

De acordo com [48], base de dados não relacionais (NoSql) não assentam nas propriedades ACID, mas sim nas propriedades BASE (*basically available*, *soft state*, *eventual consistency*), dando primazia à disponibilidade e desempenho. *Soft state* consiste na pouca consistência dos dados em todas as réplicas, e *eventual consistency*, indica que após qualquer alteração nos dados, o sistema não tem de refletir imediatamente essas alterações. Com o decorrer do tempo os dados serão consistentes em todas as réplicas, contudo a consistência não é o mais valorizado nestas propriedades.

De acordo com [49], existem cinco tipos de base de dados NoSQL: *Key-Value Store, Column-Oriented, Document-based, Graph Databases, Object-Oriented.* O primeiro tipo é um modelo muito eficiente, como o nome indica, os valores são indexados a uma chave, permitindo um utilizador obter valores especificando apenas essa chave. Quando a prioridade é obter pesquisas rápidas, onde considerámos escalabilidade em vez de consistência e diversas opções de armazenamento, NoSql é uma boa opção. *Amazon Dynamo* é um exemplo deste método [49].

Column-Oriented em vez de armazenar informações numa tabela estruturada de linhas e colunas, como numa base de dados relacional, estes tipos contêm colunas extensíveis de dados que estão relacionados (*column family*) [47]. Segundo [50], este método indexa os dados por linha, coluna e *timestamp*. As linhas e colunas são identificados por uma chave e o *timestamp* permite diferenciar as diferentes versões.

Document-based refere-se a base de dados que armazenam informação em forma de documentos, cujos formatos são XML, JSON,etc. Os documentos são indexados usando uma chave única que representa esse documento, contudo, são mais complexos em comparação com a informação que é armazenada no método Key-Value [49]. Este método oferece um ótimo desempenho e opção de escalabilidade horizontal (adicionar mais máquinas ao sistema). MongoDB e CouchDB são exemplo deste tipo de base de dados.

Graph Databases armazenam informação através de um grafo contendo nós e arestas, onde os nós atuam como sendo objetos e as arestas como ligação entre os objetos. É utiliza uma técnica *índex free adjacency*, onde cada nó aponta para o próximo nó [49]. Este método é compatível com ACID, e muito escalável.

Object Oriented é uma base de dados em que os dados são armazenados como objetos. Podemos considerar que este tipo de base de dados é uma combinação da programação orientada a objetos e os princípios de base de dados [49]. *Object Oriented* apenas é mais rápido quando a base de dados é orientada a objetos e não quando o tipo de dados é simples e sem grandes relações.

De acordo com [49], as principais vantagens de NoSQL relativamente ao SQL passa por fornecer diversos modelos de dados facilmente escaláveis, não ser necessário um administrador da base de dados, serem rápidas, eficientes e flexíveis, e conseguirem gerir falhas de hardware. No entanto, não tem só vantagens comparativamente às bases de dados relacionais, também tem desvantagens como a inexistência de uma linguagem de consulta, serem recentes e a sua manutenção ser mais difícil.

SQL e NoSQL perspetiva IoT:
Quando se trata de gerir dados heterogéneos gerados por muitos sensores e dispositivos todos conectados e integrados, é necessário garantir propriedades como escalabilidade e flexibilidade. Como referido acima, as bases de dados relacionais dão mais ênfase à propriedade de consistência dos dados, enquanto que NoSQL prioriza a escalabilidade e o desempenho.
Segundo [51], foi possível comparar três base de dados, Cassandra e MongoDB (bases de dados não relacionais), e PostgreSQL (base de dados relacional) relativamente a operações de escrita e leitura de dados. Quando apenas temos um único cliente, tanto MongoDB e PostgreSQL conseguem um desempenho superior quando realizam operações de escrita e leitura. Segundo [51], quando passamos a ter diversos clientes e múltiplos pedidos, apenas Cassandra consegue realizar um maior número de operações.

2.8 Metodologias de desenvolvimento de software

Uma das primeiras decisões que enfrentamos ao desenvolver um projeto é qual a metodologia de desenvolvimento que devemos usar. O ciclo de vida do desenvolvimento de software é um processo de construção de sistemas de software que inclui várias fases, desde a análise preliminar, até aos testes e avaliação do mesmo [52]. Existem duas metodologias utilizadas no ciclo de vida do desenvolvimento de software: Tradicionais e *Agile* [52].

Metodologias Tradicionais:
As metodologias tradicionais surgiram num contexto em que o custo de fazer alterações num software era muito alto e existia a necessidade de planear e documentar tudo antes de qualquer implementação.
Existem diversas metodologias tradicionais como *Waterfall*, o *V-Model* e *Unified Process* e embora hajam vários processos para o desenvolvimento de software essas metodologias são baseadas numa série sequencial de etapas: definição de requisitos, desenho e planeamento, implementação e testes [52]. Cada etapa só pode ser iniciada após o término da etapa anterior ser aprovada, e dificilmente existe retorno.
As etapas são:

1. A primeira fase permite determinar as características de um projeto, ou seja, perceber os requisitos, as suas restrições e o tempo que irá ser preciso para realizar as várias fases do projeto.
2. A fase de desenho e planeamento da arquitetura permite perceber problemas que possam surgir ao longo do projeto e fornece um guia para quem participa no projeto.
3. Na fase de desenvolvimento, o projeto é segmentado por diversas equipas e desenvolvido de acordo com as especificações.
4. As etapas de testes geralmente sobrepõem a fase de desenvolvimentos para garantir que os problemas das diversas etapas são resolvidos logo de início. Quando o projeto está a ser finalizado o cliente passa a participar nos testes de modo a obter feedback.

O sucesso destas metodologias depende da análise de requisitos e do conhecimento prévio adquirido. De acordo com [53], a principal preocupação dos programadores deixou de ser cumprir os requisitos que eram estabelecidos no inicio do projeto, mas sim deixar o cliente satisfeito com a entrega final. Isso só seria possível se houvessem mudanças no âmbito no projeto, modificando os requisitos e as tecnologias anteriormente estabelecidas.

Metodologia Ágil:
Em 2001 quando 17 especialistas se juntaram, para discutir sobre processos de desenvolvimento de software, surgiu uma nova metodologia designada de desenvolvimento ágil [54]. Esta metodologia teve um enorme impacto na forma como o software é desenvolvido, pois permite a existência métodos para lidar rapidamente com diversas alterações, prazos e necessidades ao longo do projeto. Esta metodologia assenta em quatro valores [55]:

1. Interações entre indivíduos, processos e ferramentas
2. Software funcional de acordo com a documentação
3. Colaboração do cliente na negociação do contrato
4. Capacidade de responder as alterações seguindo um plano

Interações entre indivíduos permite a troca de informações, minimizar a documentação necessária e se for preciso realizar alterações. De acordo com [55], existem 6 métodos ágeis: *Crystal methodologies, Dynamic software Development* (DSDM), *Feature-driven Development, Lean software development, Scrum, Extreme programing*(XP). Cada um destes métodos aborda diferentes perspetivas da desta metodologia. Por exemplo, o DSDM exige que se contruam protótipos para perceber as áreas de maior risco e o Scrum dá enfase às reuniões diárias onde se discute como está o desenvolvimento das tarefas [53]. Segundo [52], a principal prioridade dos métodos ágeis é o retorno do investimento. A grande diferença com os métodos tradicionais, é a capacidade de entregar resultados com baixo custo, com rapidez e com requisitos que ainda poderão ser alterados. Os métodos ágeis dão enfase ao trabalho em equipa, e à rápida resposta à mudança, enquanto que os convencionais priorizam contratos, planos e documentos de trabalho.

2.9 Exemplos de soluções reais usando AR e IoT

Nesta secção são referidos exemplos de organizações que utilizam AR e IoT nos seus modelos de negócio.

CAT Liveshare:
Segundo [56], a Caterpillar *Liveshare* (CAT – Caterpillar) é uma nova ferramenta de colaboração que permite aos técnicos e especialistas remotos se liguem por meio de vídeo e áudio para resolver um problema em tempo real usando realidade aumentada. Essa plataforma de vídeo baseada em realidade aumentada, utiliza voz, animações 3D, anotações onde os utilizadores podem desenhar, destacar e colocar setas sobre objetos do mundo real e partilhar de tela [57]. É utilizado transmissão de vídeo por meio de Wi-Fi ou telemóvel. Quando essa cobertura de wi-fi ou móvel for baixa, os técnicos podem enviar imagens instantâneas em vez de transmissão por vídeo, tornando assim possível reduzir a largura de banda. A CAT desenvolveu essa plataforma com o objetivo de diminuir o tempo de resolução de um problema, aumentar as receitas de serviços e aumentar a satisfação do cliente.

Bosch Automotive Service Solutions:

Em 2014, a *Bosch Automotive Service Solutions*, introduziu uma plataforma designada de plataforma comum de realidade aumentada (CAP - *Platform Common Augmented Reality*), que fornece um sistema de multi-plataforma capaz de criar aplicações AR.

O CAP da Bosch permite a integração de conteúdo visual e digital no processo de criação [56], através de imagens ao vivo e de uma base de dados na qual se extrai conteúdo correspondente a aplicação AR. Essa sobreposição visual é feita através de modelos 3D, fotos, textos (informações, explicações) e *tracking* de objetos. Relativamente à manutenção, assistência técnica ou reparação, o CAP é utilizado para visualizar cabos ou componentes ocultos e instruções de reparação que permitem visualizar as etapas necessárias de trabalho e as respetivas ferramentas. Também é possível verificar o estado das peças com informações em tempo real. O CAP permite a ampliação e rotação de objetos e oferece informações adicionais como vídeos de treinamento.

Os principais objetivos desta plataforma é reduzir o tempo com decisões mais rápidas e aumentando o processo de montagem, economizar custos e reduzir as taxas de erros.

Airbus:

A empresa Airbus é a maior empresa aeronáutica da Europa, sendo responsável pela montagem de aeronaves e helicópteros em diversos locais do mundo. Para criar um novo processo de produção de aeronaves através de ferramentas digitais, em 2009, foi criado o MiRA (*Mixed reality Application*) [58]. Esta aplicação aumenta a produtividade nas linhas de produção usando realidade aumentada para visualizar peças e detetar erros [58]. É utilizado um tablet, sensores e um software especialmente desenvolvido para esta aplicação, de forma a reduzir o tempo necessário para verificar, recuperar e montar peças que possam estar danificadas. Com essa aplicação foi possível reduzir 300 horas de suporte na fuselagem (isto é, carcaça principal da aeronave) para 60 horas.

DHL:

A DHL é uma empresa do setor de logística internacional que está a tentar implementar realidade aumentada no seu setor, nomeadamente em operações de armazenamento, otimização no transporte, serviço de entregas e serviços de montagem/desmontagem.

Segundo [59], qualquer abordagem no papel é lenta e propensa a erros, para isso, foi criado um software que permite o reconhecimento de objetos em tempo real, leitura de código de barras, navegação interna e integração de informações com um sistema de gestão de armazém. Com este sistema, cada trabalhador pode visualizar a melhor rota até ao locar onde o objeto se encontra, utilizar código de barras para leituras automáticas e reconhecimento do local. Também é possível visualizar um processo de armazenamento, de modo a perceber como o armazém se encontra organizado.

A realidade aumentada pode otimizar o transporte de carga, através da verificação da mesma, permitindo saber se existe capacidade num camião de a transportar. Outra funcionalidade para otimizar o transporte é a utilização de AR no transito, permitindo ao motorista visualizar um sistema de navegação exterior com informações do trânsito em tempo real.

No serviço de entregas ao usar realidade aumentada seria possível perceber qual o motorista adequado a uma determinada carga, calculando e visualizando o espaço livre em cada carregamento.

A DHL é responsável por armazenar componentes da marca Audi, mas também por montar os seus componentes. Para otimizar esse processo de montagem e reduzir as taxas de erros, um sistema de realidade aumentada pode oferecer uma maneira intuitiva de visualizar esse mesmo processo, de forma a fornecer instruções a cada etapa de trabalho.

Thyssenkrupp *Elevator*:

A Thyssenkrupp é uma das principais empresas de construção e manutenção de elevadores e faz uso de AR e IoT nos seus equipamentos. O seu principal objetivo é reduzir significativamente os custos de manutenção e garantir que os seus elevadores funcionem corretamente. Trabalhando ao lado da Microsoft, a empresa conseguiu interligar os sensores dos seus elevadores à nuvem do *Azure* [60]. Através deste sistema eles conseguem capturar dados como temperatura do motor, alinhamento do eixo, velocidade e funcionamento da porta. Além disso, quando os técnicos se dirigem ao local, é possível visualizar instruções de reparação através de realidade aumentada.

Capítulo 3

O Planeamento

3.1 Metodologia de desenvolvimento

De forma a integrar os seus clientes no processo de desenvolvimento, a Accenture adota uma estratégia baseada na metodologia *Agile*, designada de *Scaled Agile Framework* (SAFe) com DevOps. Assim sendo, devido ao uso desta metodologia entre diversos elementos da Accenture, foi adotada a mesmo estratégia com o estagiário de forma a facilitar a integração com a metodologia. A metodologia *agile* envolve uma interação contínua entre os diversos elementos da equipa e permite a implementação de diversas funcionalidades num intervalo curto. No entanto, durante o processo de desenvolvimento esteve em falta uma equipa, elemento importante nesta metodologia. Devido a ausência da equipa, apenas o estagiário participou no desenvolvimento do projeto, no entanto, sempre sob supervisão dos orientadores através de reuniões periódicas.

3.2 Reuniões

Ao longo do desenvolvimento do projeto ocorreram reuniões periódicas com o orientador da empresa e o orientador da FCUL. Essas reuniões tiveram como objetivo garantir o cumprimento dos requisitos e de um ponto de vista mais académico, assegurar que os prazos para a entregas dos relatórios eram cumpridos. Numa fase inicial do projeto, foi possível participar em reuniões com diversos gestores da empresa responsáveis por diversas áreas, permitindo assim definir melhor os requisitos necessários que o projeto deveria conter. Na fase final do estágio as reuniões consistiram em apresentar o projeto de forma a obter um *feedback* relacionado com as funcionalidades desenvolvidas.

3.3 Calendarização do projeto

O planeamento do projeto foi concebido considerando o período do estágio de 9 meses, tendo sido segmentado em duas fases. A primeira fase foi dedicada essencialmente à investigação sobre o problema proposto de forma a perceber quais seriam possíveis soluções e tecnologias tendo em conta o âmbito do projeto. A segunda fase consistiu em definir como seria o sistema, a sua arquitetura e a implementação do projeto. Na Figura 3.1 está representado o mapa de Gantt de todo o planeamento.

- **15 de Outubro – 23 de Novembro:**
  -Reuniões de levantamento de requisitos
  -Análise funcional e arquitetural
  -Pesquisa e análise do Estado da Arte

- **12 de Novembro – 14 de Dezembro:**
  -Escrita do relatório preliminar

- **14 de Dezembro – 25 de Janeiro**
  -Desenvolvimento do *Minimum Viable Product**

- **28 de Janeiro – 31 de Maio**
  -Desenvolvimento da solução mobile
  -Testes do sistema e levantamento de feedback
  -Análise de resultados

- **3 de Junho – 19 de Julho**
  -Escrita da dissertação

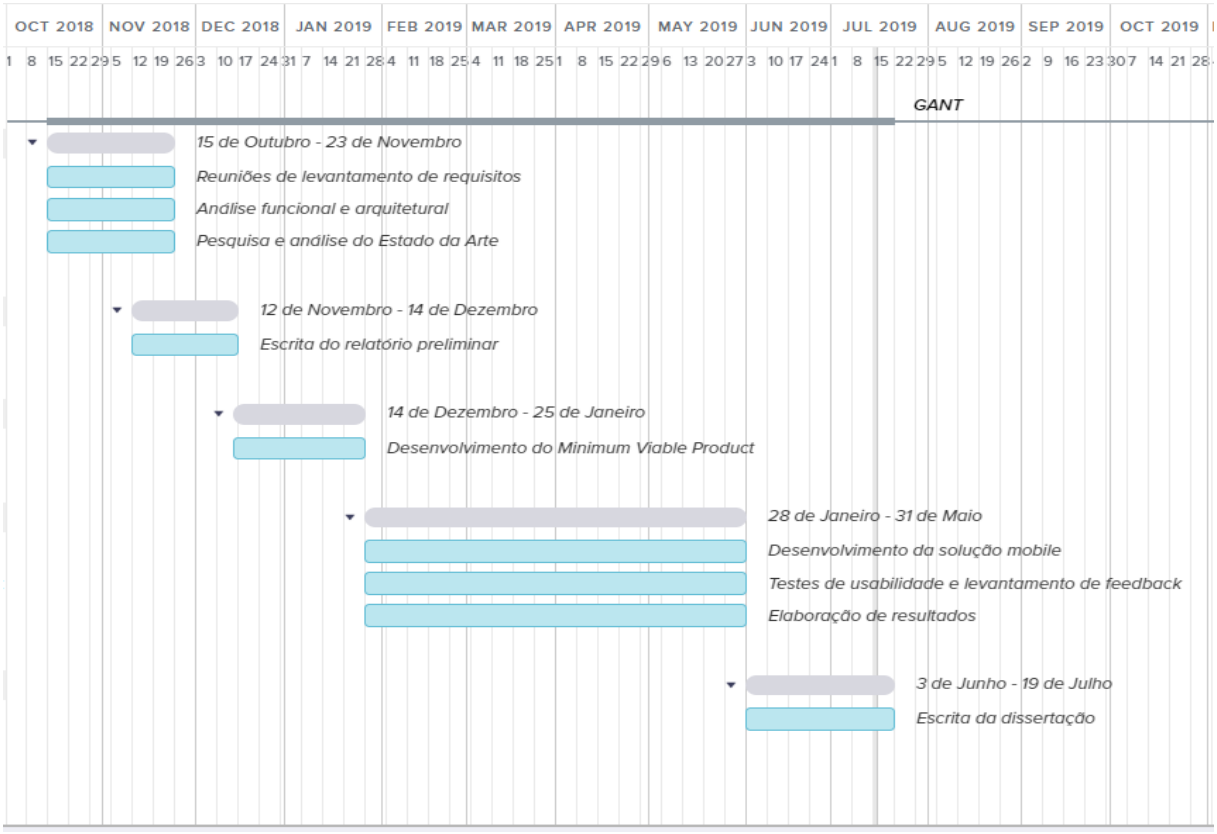

Figura 3.1-Mapa do Gantt

De referir que grande parte do planeamento foi cumprido, contudo existiram pequenos problemas que acabaram por atrasar o projeto, nomeadamente durante o desenvolvimento da solução móvel. Numa fase inicial apesar de terem sido estudadas as tecnologias a serem utilizadas existiram problemas técnicos que levaram ao atraso da construção da solução. Contudo, não houve necessidade de alterar o planeamento, pois não foram problemas que justificassem a alteração de prazos. Esses problemas técnicos são detalhados e explicados no subcapítulo 6.5 .

Capítulo 4 Levantamento de Requisitos

Durante as várias reuniões com os gestores da Accenture, foi possível determinar os requisitos que o sistema deve obedecer. Por isso, este capítulo tem como objetivo descrever os requisitos da solução desenvolvida. É um processo essencial pois permite descrever características e funcionalidades que o sistema deve ter. Seguidamente são enunciados os requisitos funcionais, não funcionais e algumas *user stories* relevantes para o desenvolvimento da solução.

4.1 Requisitos funcionais

O levantamento de requisitos permite que um utilizador possa ter a perceção de um problema ou de uma necessidade existente numa determinada área. A qualidade de um produto ou processo está diretamente relacionada com a qualidade dos requisitos especificados. Este processo começou com análise do domínio da aplicação, tentando perceber os conceitos e desafios de *Field Force Management.*
Durante as reuniões foi possível perceber que apesar de existir a vertente do cliente e do técnico, o verdadeiro impacto de AR e IoT está na vertente do técnico. Os requisitos funcionais aqui apresentados foram definidos após as diversas reuniões entre o estagiário e o orientador.

Requisitos Funcionais:
No âmbito de realidade aumentada e IoT, após diversas pesquisas e reuniões sobre os requisitos a ter em conta, foi possível determinar os requisitos funcionais a considerar. A Tabela 4.1 contém todos os requisitos que a aplicação deve conter.

RF01	O sistema deve permitir que técnico consiga visualizar anomalias através dos valores de sensores embutidos no equipamento.
RF02	O sistema deve permitir que o técnico consiga digitalizar o QR *code* de um equipamento
RF03	O sistema de AR deve permitir que o técnico consiga visualizar o modelo 3D de um equipamento
RF04	O sistema de AR deve permitir que o técnico tenha acesso a relatórios de manutenção
RF05	O sistema de AR deve fornecer instruções de reparação, nomeadamente os passos a seguir de um processo de montagem/desmontagem ou instalação de componentes
RF06	O sistema deve permitir ao técnico gerar um relatório de manutenção
RF07	O sistema deve permitir que apenas técnicos autenticados possam aceder às informações.

Tabela 4.1- Requisitos funcionais de AR e IoT na aplicação

4.2 Requisitos não-funcionais

Requisitos não funcionais:
Os requisitos não funcionais definem qualidades ou atributos do sistema, e não funcionalidades diretas de um sistema. Na Tabela 4.2 foram mencionados 2 requisitos que devem ser considerados no desenvolvimento da aplicação.

RFN01	Desempenho	O sistema deve conter recursos para processar em paralelo grande quantidade de dados, possibilitando o seu armazenamento.
RFN02	Escalabilidade	O sistema deve ser capaz de suportar um aumento substancial de dados a processar.
RFN03	Interoperabilidade	O sistema deve estar preparado para integrar diversos dispositivos com diversos formatos e padrões.
RFN04	Segurança	O sistema deve garantir que apenas o utilizador autenticado tem acesso às informações.
RFN05	Confidencialidade	O sistema deve garantir também a confidencialidade dos dados trocados entre a aplicação e a base de dados.
RFN06	Fiabilidade	O sistema deve funcionar de acordo com as especificações sempre que necessário.
RFN07	Usabilidade	O sistema deve conter uma interface intuitiva e de fácil compreensão, de modo a não confundir o utilizador.
RFN08	Compatibilidade	O sistema deve funcionar em diversos dispositivos móveis com o sistema operativo Android

Tabela 4.2-Requisitos não funcionais da aplicação

4.3 User stories

Uma *user story* consiste numa descrição curta e informal, de uma ou mais funcionalidades do sistema. As *user stories* são descritas do ponto de vista do utilizador e o seu foco está no "porque" e "como" esse utilizador interage com o sistema, podendo por vezes ser visto como requisitos que o sistema deve ter. No entanto, não devem ser confundidas com requisitos, visto que estes entram em mais detalhes e servem para orientar uma equipa como o *software* deve funcionar, enquanto que *user stories* são descrições informais claras e simples das funcionalidades. Existe vários exemplos de como escrever uma *user stories* mas para este caso será utilizado o seguinte formato:

Como <**tipo de utilizador**>, pretendo <**objetivo**> de forma a que <**motivo**>

No caso da solução desenvolvida, o tipo de utilizar será sempre o mesmo, visto que foi definido anteriormente que seria para a vertente do técnico. O tipo de utilizador será um técnico de uma empresa que fornece serviços de manutenção e reparação de equipamentos. Por isso, o campo <tipo de utilizador> será sempre substituído por técnico. Após as reuniões de levantamento de requisitos, na Tabela 4.1 estão descritas as *user stories* relevantes para a solução:

ID	Problema	Descrição
US1	Modelo 3D do equipamento	**Como** técnico, **pretendo** visualizar o modelo 3D do equipamento **de forma a que** consiga ver a localização exata dos problemas que um equipamento pode ter.
US2	Visualizar relatórios de manutenção	**Como** técnico, **pretendo** ter acesso aos relatórios de manutenção gerados **de forma a que** consiga perceber se houve anteriormente gerados numa determinada peça.
US3	Gerar relatórios de manutenção	**Como** técnico, **pretendo** gerar um relatório de manutenção **de forma a que** qualquer técnico tenha acesso a esse relatório
US4	Visualizar instruções de reparação	**Como** técnico, **pretendo** visualizar o processo de reparação do equipamento através de instruções 3D **de forma a que** consiga solucionar o problema o mais rapidamente possível

Tabela 4.3- *User stories* da aplicação

Capítulo 5 Desenho da solução

5.1 Visão geral

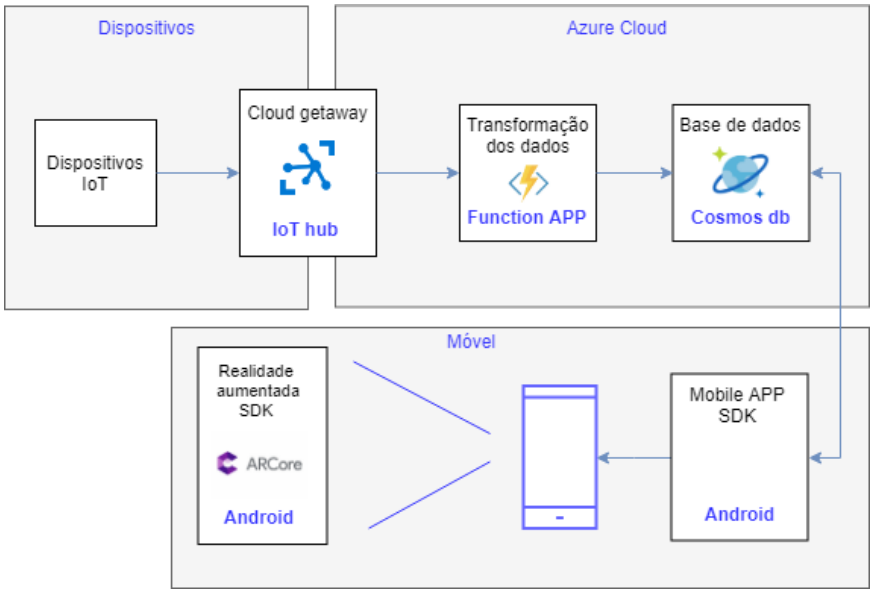

Figura 5.1- Visão geral do sistema

A vista simplificada do sistema mostra a ligação entre as diversas componentes de IoT e AR de forma a ser possível ter uma visão geral do sistema. Como podemos observar na Figura 5.1 , a vista geral da solução pode ser repartida em 3 blocos: dispositivos, nuvem Azure e móvel. O bloco dos dispositivos usa componentes do Azure PaaS para criar soluções de IoT.

Os dispositivos IoT contém sensores ligados diretamente numa máquina e têm a capacidade de se registar com segurança na nuvem de forma a enviar e receber dados. *Cloud gateway* é responsável por receber a informação proveniente dos dispositivos. O IoT *hub* é um serviço da nuvem que recebe eventos gerados pelos dispositivos, agindo como intermediário de mensagens entre os dispositivos e serviços de *back-end*. A transformação de dados manipula e altera o fluxo de telemetria, por exemplo, converter diversos tipos dados em JSON, que neste caso específico, serve para extrair a informação necessária. As funções do Azure possuem integração com o IoT *hub* e o Cosmos DB. O Cosmos DB é uma base de dados geralmente classificada como *no-sql* que recebe a informação do IoT *hub* e seguidamente cria uma coleção que contém documentos com essa informação. Cada dispositivo contém um identificador único de forma a ser possível armazenar dados conforme esse id.

Na solução móvel para obter os dados é utilizado um SDK android para a API SQL do Azure *Cosmos DB*. O SDK utilizado serve como interface com a base de dados que suporta persistência offline de leitura e escrita de documentos. Quando é necessário ter acesso aos documentos armazenados na base de dados, um utilizador através de *queries* SQL consegue obter a informação que pretende. Para além desse *software*, na componente móvel é utilizado outro SDK designado de *ArCore*. Esse SDK tem o objetivo de visualizar através de AR o objeto 3D que irá ser reparado. No entanto, também é possível

visualizar nesse mesmo objeto 3D os seus respetivos problemas que estão armazenados nos documentos provenientes do Azure Cosmos DB.

5.2 Padrão Arquitetural do sistema

Os padrões arquiteturais apresentam uma visão simplificada e abstrata de um subconjunto de implementações de um sistema IoT. São exemplos e referencias para a conceptualização de um sistema no mundo real, onde os arquitetos podem reconhecer os recursos necessários de um sistema.

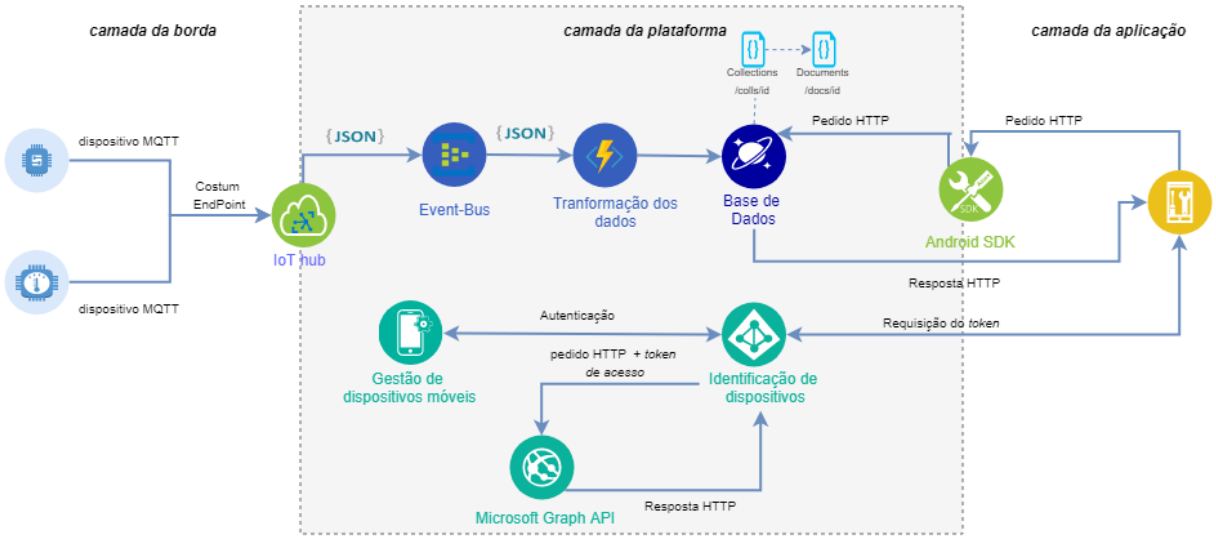

Figura 5.2- Arquitetura de três camadas do sistema

A Figura 5.2 apresenta uma arquitetura em três camadas, abrangendo a camada da borda, a camada da plataforma e camada da aplicação, com base no que é descrito na *Industrial Internet Consortium Reference Architecture (IIRA)* em [61] .
A camada da borda recolhe dados dos dispositivos utilizados para detetar eventos e transmitir pela rede até chegar ao IoT *hub*. Para a troca de mensagens entre os dispositivos IoT foi utilizado o protocolo MQTT, que é um protocolo de mensagens projetado para comunicações M2M.
A camada da plataforma são serviços da nuvem, responsáveis por receber, processar e analisar o fluxo de dados provenientes da camada da borda. Podemos segmentar esta camada em duas nomeadamente de transformação dos dados e de autenticação. O segmento de transformação dos dados fornece serviços de transformação de dados antes de eles serem armazenados, pois quando os dispositivos enviam informação para a nuvem, podem ter diversos formatos. O objetivo desses serviços na nuvem é extrair a informação relevante desses formatos de dados antes de os armazenar na base de dados. Como podemos visualizar na arquitetura, quando o IoT *hub* recebe algum evento é utilizado uma função de tratamento dos dados que extrai a informação e armazena num documento na base de dados.
O segmento de autenticação permite integrar um serviço de autenticação e autorização entre a aplicação e o utilizador. O processo de autenticação é feito através dos serviços da nuvem, que são responsáveis por gerar *tokens* de sessão após verificar se um utilizador tem acesso a um determinado tipo de recursos, explicado no subcapítulo 5.5 . Após a autenticação ser verificada é possível obter as informações do utilizador, através uma API REST da Microsoft Graph API.

A camada de aplicação implementa aplicações especificas do sistema, recebendo informações que foram solicitadas. A aplicação apenas pode aceder aos recursos e as informações após autenticação no sistema e posteriormente utiliza um SDK responsável por comunicar com os serviços da nuvem, através de pedidos HTTP. Esta camada recebe informação após ter sido transformada pela camada da plataforma e posteriormente armazenada.
O padrão arquitetural de 3 camadas combinas os principais componentes de um sistema IoT, que na perspetiva desta arquitetura, a camada da plataforma implementa a maior parte do domínio de informações e operações, contudo, isso pode variar consoante as especificidades dos casos e dos requisitos do sistema. Podemos utilizar ou transportar alguns serviços da camada da plataforma, para junto da camada da borda, habilitando assim a computação da borda inteligente como descrito no trabalho futuro (subcapítulo 8.3).

5.3 Padrão arquitetural da aplicação móvel

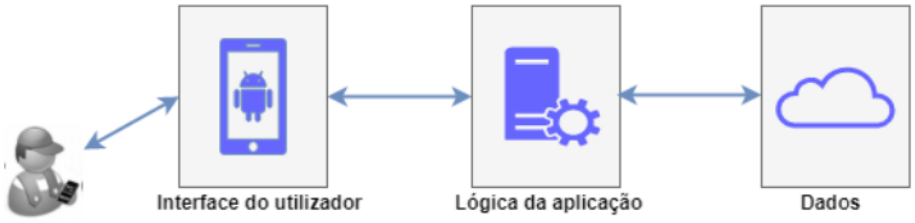

Figura 5.3-Camadas da aplicação móvel

Como podemos observar na Figura 5.3, a solução móvel está dividida em 3 camadas: (i) a camada de apresentação; (ii) a camada da aplicação; (iii) a camada de acesso aos dados. Esta solução foi construída com base no padrão arquitetura MVC, por isso podemos classificar a *Figura 5.3* também como camadas *View, Controller e Model*, respetivamente. A interface do utilizador, como o próprio nome indica, é a camada que faz a interação do utilizador com a aplicação. Nesta camada, está situada todos os ficheiros .XML responsáveis pela apresentação do conteúdo.
A lógica da aplicação é a camada responsável por gerir a comunicação entre a interface do utilizador e base de dados. Funciona como um intermediário entre a aplicação móvel e a base de dados, fazendo *queries* sempre que necessário obter ou alterar dados na base de dados. No caso desta solução, temos a lógica de subdividida pelos *packages*: (i) *Animations;* (ii) *Augmented_Reality;* (iii) Reports (iv) *Model*. Contudo, isso será descrito no capítulo 6.3 (Estrutura da aplicação).
A camada de acesso aos dados corresponde as ligações realizadas com a base de dados onde está armazenado a informação. Na solução móvel corresponde a classes responsáveis por fazer *queries* ao Cosmos DB através de um SDK, que serve como interface com a base de dados. Sempre que houver determinados eventos na camada de aplicação, essa comunica com a camada de acesso aos dados que posteriormente comunica com a base de dados. Por vezes a camada de aplicação por fazer também comunicação com a base de dados devido às tarefas assíncronas que a aplicação está a realizar. Este padrão fio adotado devido ao seu grande uso em aplicações android e facilidade de implementação.

5.4 Tecnologias e linguagens utilizadas

A implementação da solução foi feita através das seguintes tecnologias:

- Android Studio 3.4.1: IDE para o desenvolvimento de aplicações para Android, recorrendo a linguagem Java e uma interface com o utilizador baseado em XML.

- Visual Studio Code: IDE desenvolvido pela Microsoft que permite usar uma variedade de linguagens de programação. Pode usar *plug-ins* disponíveis no seu repositório. A extensão do arduino no VS Code permite executar/compilar código diretamente no dispositivo.

- Blender: Conjunto de ferramentas de software de computação gráfica 3D de código aberto que permite modelar objetos 3D, animações, efeitos visuais, etc [62]. Blender é uma ferramenta multi-plataforma que funciona em computadores Linux, Windows e Macintosh.

- *ArCore Sceneform*: API para *rendering* de modelos 3D usando Java. Com recurso a um *plug-in* no Android Studio é possível importar, visualizar e criar recursos 3D para ser usado em AR. A única limitação é requerer uma versão de android igual ou superior à versão 3.1.

- Azure Function: *Serverless* é um modelo de execução onde o fornecedor de nuvem executa código com determinados recursos que vão ser alocados dinamicamente. Azure Function é um serviço de computação *serveless*, que através de linguagens como C#, JavaScript, Java, F e das bibliotecas .Net, permite executar *scripts* em resposta a diversos eventos. A integração entre o Cosmos DB e o Azure Function permite criar *triggers* de base de dados e vincular dados de saída diretamente para a base de dados.

- Cosmos DB API SQL: O Cosmos DB é uma base de dados geralmente classificada como no-sql, no entanto, permite a utilização de uma API SQL. Permite fazer *queries* através de comandos como "*Select*" ou "*Insert*", onde é possível armazenar, manipular e verificar dados inseridos na base de dados, que neste caso serão documentos com informações sobre os valores dos sensores.

- Git: O Git é um sistema de controle de versões distribuído e de código aberto de ficheiros. Possibilita coordenar o trabalho de uma equipa de desenvolvimento. Contém um controlo de versões, onde regista as mudanças feitas num ficheiro ao longo do tempo, possibilitando recuperar versões antigas.

5.5 Segurança

No âmbito de projeto existem algumas considerações relacionadas com a segurança e confiabilidade dos dados devem ser tidos em conta. Ao construir a solução é necessário garantir a proteção da base de dados, a confidencialidade das mensagens trocadas entre a aplicação e o serviço, e por último, que apenas o técnico autenticado pode ter acesso a aplicação.
Nesta secção serão apresentadas quais as considerações de segurança relevantes, procurando satisfazer os requisitos não funcionais RNF04 e RNF05.

Autenticação do utilizador:

Todos os sistemas que têm de manter sessões ou troca de informação confidencial entre o sistema e o utilizador, exigem mecanismos de autenticação. A autenticação é um processo que permite confirmar a identidade de um utilizador, para garantir que o mesmo utilizador tem acesso à informação. A finalidade principal consiste em impedir que pessoas não autorizadas tenham acesso ao sistema.
No âmbito do projeto e de forma a simplificar a autenticação foi utilizado a plataforma de identidade da Microsoft. Essa plataforma é responsável por verificar a identidade dos utilizadores e aplicações, em que caso a autenticação ser bem-sucedida emite um *token* de segurança durante um determinado período. A aplicação nativa obtém o *token* de acesso usando o protocolo OAuth2.0 e é então enviado uma solicitação para a API Web, que autoriza que o utilizador tenha acesso aos recursos.

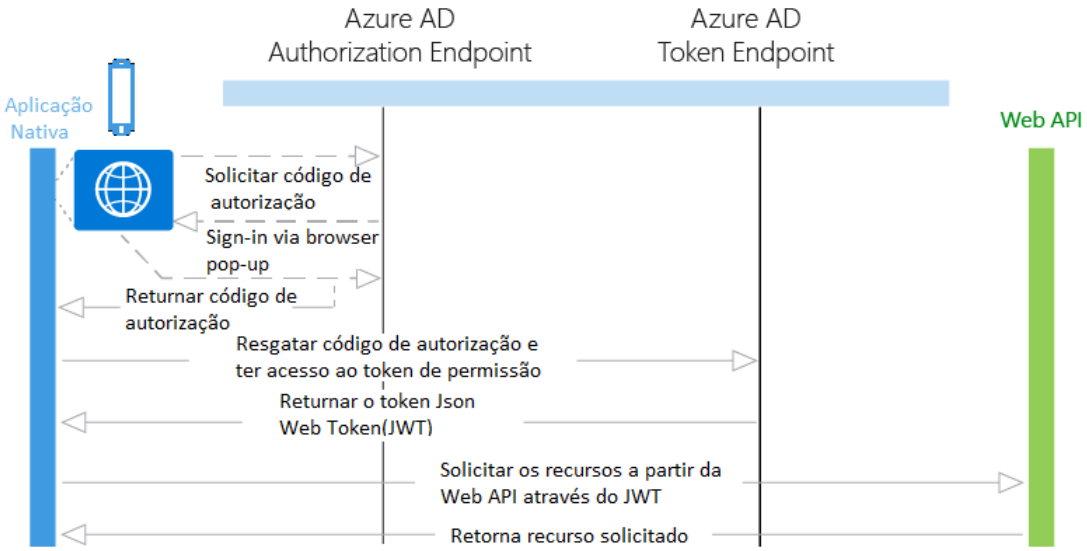

Figura 5.4- Diagrama de autenticação do Azure AD, adaptado de [63]

A Figura 5.4 ilustra o funcionamento de autenticação do Azure Active Directory.
Segundo [63], quando um utilizador deseja fazer login, através de um *Browser*, a aplicação nativa faz uma solicitação ao *end-point* do Azure AD, onde inclui o ID e o URI da aplicação, conforme mostrado no portal do Azure. O Azure AD autentica o utilizador e emite uma resposta em forma de código de autorização para o URI da aplicação do cliente. Quando o Azure emite essa resposta, a aplicação é interrompida para obter o código de autorização da resposta. Seguidamente, a aplicação envia um pedido ao *Azure AD's token endpoint* que inclui o código de autorização, detalhes sobre a aplicação (ID e URI) e o recurso desejado. Após o código de autorização ser valido o Azure AD retorna dois *tokens*: um *token* de acesso ao JWT e um *token* de atualização. Através de HTTPS, a aplicação faz uso desses *tokens* de acesso para aceder aos recursos que solicitou.

Azure Cosmos DB:

O objetivo da implementar segurança em base de dados, é proteger a última de ameaças. Essas ameaças representam um risco grande para a integridade e confiabilidade dos dados. Além disso, a segurança das bases de dados permite limitar o acesso de utilizadores, reduzindo apenas o acesso aos dados quem está autorizado. Segundo [64], existem 3 ameaças que devemos considerar: modificação não autorizada, divulgação não autorizada e perda de disponibilidade. A modificação não autorizada é quando existe uma alteração dos dados, por motivos de sabotagem, que não deveria ser permitido. A divulgação não

autorizada como o próprio nome indica, é quando informações privadas são divulgadas. Perda de disponibilidade do serviço, é quando uma base de dados não está disponível.
A segurança dos dados é uma responsabilidade repartida entre o cliente e o fornecedor de base de dados. Dependendo desse fornecedor, a responsabilidade pela segurança pode ser variável tanto do lado do cliente como da base de dados. O Azure Cosmos DB é um serviço de base de dados na nuvem PaaS (*Plataform as a Service*) capaz de lidar com as ameaças acima mencionadas. O Azure Cosmos DB oferece uma distribuição global permitindo replicar os dados em qualquer centro de dados mundial conseguindo assim proteção contra perda de disponibilidade. Para evitar que pessoas não autorizadas tenham acesso aos dados controlar o acesso aos recursos da base de dados, o Azure usa dois tipos de chaves (primária e secundária) para autenticar os utilizadores [65]. Como podemos observar na Figura 5.5, as chaves podem ser obtidas diretamente no portal do Azure e só quem tem essas chaves pode ter acesso aos recursos da base de dados. Caso exista algum problema com as chaves, a qualquer momento o administrador da base de dados pode alterar essas chaves.

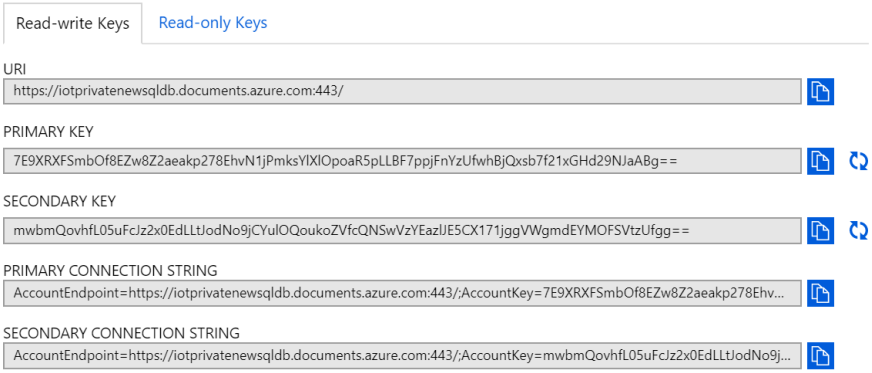

Figura 5.5- Chave primária e secundária

HTTPS:

Secure Sockets Layer (SSL) é um protocolo projetado para fornecer segurança entre comunicações entre sistemas na internet. O SSL foi adotado em comunicações entre sistemas que normalmente envolvem um cliente e um servidor, e de forma a evitar a interceção de dados nessas comunicações. Para garantir essa segurança, o SSL usa um sistema de criptografia que usa duas chaves para encriptar os dados, uma chave pública conhecida por todos e uma chave secreta conhecida apenas pelo destinatário da mensagem. O HTTPS é a versão segura do HTTP com SSL/TLS, originando assim o HTTPS. O HTTPS autentica os *end-points* e fornece confidencialidade nas comunicações, prevenindo assim que ataques como por exemplo, *Man-in-the-Middle*, tenham sucesso.
A API REST do Azure Cosmos DB fornece acesso aos recursos da base de dados para criar, consultar e excluir documentos através de solicitações HTTPS, permitindo cifrar o fluxo de dados entre o servidor e seu aplicativo [66].

Capítulo 6 Implementação

Neste capítulo são apresentadas as várias fases do desenvolvimento da solução proposta neste projeto. Este capítulo está divido em 6 partes: (i) sistema IoT, (ii) AR, (iii) estrutura da aplicação, (iv) funcionalidades, (v) principais obstáculos. Inicialmente é apresentado o sistema IoT proposto na arquitetura, desde a camada dos dispositivos até aos serviços da nuvem. Na componente de AR é descrito o SDK utilizado na modelação dos objetos 3D. Seguidamente é explicado a estrutura da aplicação seguindo o modelo MVC e posteriormente as funcionalidades que foram implementadas. Por fim, é descrito os principais obstáculos durante o processo de desenvolvimento.

6.1 Internet of Things

6.1.1 Dispositivos IoT

A utilização de dispositivos reais numa solução baseada em IoT é uma mais valia pois torna possível trabalhar sobre dados de contexto real. O foco nesta solução seria arranjar um ou mais dispositivos que contivessem vários sensores integrados para facilitar o uso de dados reais sem ser necessário grandes configurações nos dispositivos. Para isso foi utilizado um equipamento designado de MxChip IoT, com o propósito de desenvolver soluções de IoT tirando partido dos serviços do Azure. Como podemos observar na Figura 6.1, o MxChip tem integrado módulos e sensores como temperatura, pressão, humidade, entre outros. O EMW3166 é um módulo Wi-Fi de baixo consumo de energia desenvolvido pela MXCHIP que permite ligar ao Azure com rapidez e segurança.
No Visual Studio Code foi usado a extensão do arduíno para permitir executar/compilar código diretamente no dispositivo, possibilitando configurar o MxChip para enviar um determinado tipo de dados para o Azure, que neste caso seriam dados de temperatura e pressão. A segurança do dispositivo também está implícita quando configuramos o dispositivo. O MXChip contém um chip, onde na imagem abaixo corresponde ao *Security IC* para garantir que dados como a chave de ligação para aceder aos serviços do Azure são armazenados com segurança. Para além disso, permite que os dados de comunicação sejam encriptados através de uma chave simétrica. Para garantir que um dispositivo autenticado pode enviar dados diretamente para o IoT *hub* é necessário introduzir a chave primária e secundária no *VS Code* antes de executar o código no dispositivo.

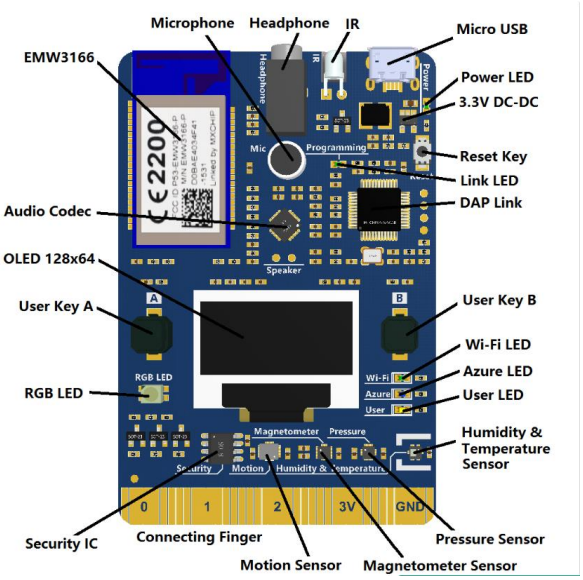

Figura 6.1- MXChip IoT

6.1.2 Cloud gateway

O IoT *hub* é um serviço da nuvem fornecido pela Microsoft Azure, que atua como um *hub* central de mensagens para a comunicação bidirecional entre aplicações IoT e os seus dispositivos na rede. É um serviço que permite criar soluções de IoT com comunicações de confiança entre os dispositivos e serviços *back-end* na nuvem, pois requer que cada dispositivo seja autenticado como descrito no capítulo 5.5 (segurança). Este serviço consegue responder às características impostas num sistema IoT. Suporta vários padrões de sistemas de mensagens bem como métodos para controlar os seus dipositivos a partir da *nuvem*. O IoT *hub* foi projetado para conseguir ter milhões de dispositivos ligados simultaneamente e responder a milhares de eventos por segundo. Para além, disso também é possível integrar o IoT *hub* a outros serviços de Azure permitindo criar novas soluções. De acordo com estas funcionalidades, podemos afirmar que características como escalabilidade, interoperabilidade e segurança são asseguradas pelo sistema. Na Figura 6.2 é possível observar um IoT *hub* em funcionamento, com 8 dispositivos autenticados.

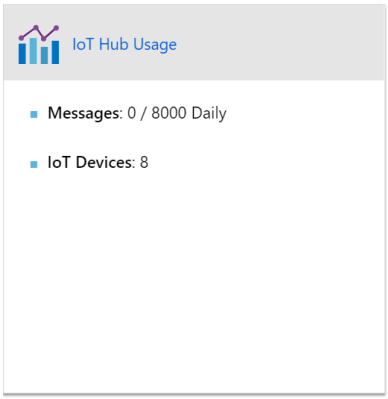

Figura 6.2- Exemplo do número de dispositivos do sistema

6.1.3 Transformação dos dados

As funções do Azure permitem criar pequenas funções reutilizáveis para determinado tipo de tarefas, como por exemplo, responder a alterações na base de dados. São funções orientadas a eventos e podem "ouvir" alterações que acontecem na base de dados, permitindo o seu funcionamento em paralelo. Relativamente a esta solução, foi criada uma função Azure desenvolvida com C# e bibliotecas .NET, com o objetivo de transformar os dados dessa mensagem retendo apenas o que é importante sempre que houver uma mensagem enviada de um dispositivo para o IoT *hub*.

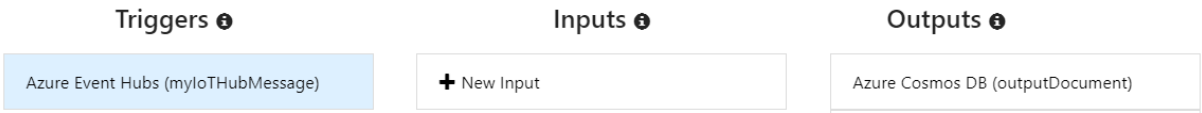

Figura 6.3- *Trigger* da função do Azure

Na Figura 6.3 é possível observar uma como uma função do Azure tem um *trigger* integrado com o IoT *hub* acionando a função sempre que é gerado um evento. Devido a integração entre o Cosmos DB e a função Azure, após tratar a informação podemos definir as variáveis do documento diretamente na função e posteriormente enviar para a base de dados.

```
public static void Run(string myIoTHubMessage,out object outputDocument, ILogger log)
{
    log.LogInformation($"C# IoT Hub trigger function processed a message: {myIoTHubMessage}");
```

Figura 6.4- Parâmetros de entrada da função do Azure

Na Figura 6.4 temos os parâmetros de entrada quando um evento é gerado. Quando o *trigger* é accionado a string *myIoTHubMessage* que contém o JSON do evento é usada para extrair informação como o nome do dispositivo, o seu id, o alerta que foi gerado e o seu valor.

```
2019-06-27T12:16:26.341 [Information] C# IoT Hub trigger function processed a message: {
    "deviceId": "MXChip",
    "temperature": 24.600000
}
```

Figura 6.5- Exemplo de uma mensagem recebida no IoT *hub*

Figura 6.6- Exemplo de um documento da base de dados

A Figura 6.5 representa uma mensagem Json que é enviada pelo dispositivo caso aconteça um alerta. Como podemos observar, as mensagens contêm o id do dispositivo e o valor do alerta sendo neste caso a temperatura do dispositivo que atingiu os 24ºC. Após a informação ser extraída e tratada na própria função é posteriormente criado um documento com essa informação, acrescentado a data do alerta. A Figura 6.6 é um exemplo de um documento criado após o evento ser gerado. Através desta função é possível visualizar o fluxo de dados e construir regras para esses dados de forma controlar o que é guardado na base de dados.

6.1.4 Base de dados

Os dados dos dispositivos IoT tem características como (i) pequenos e imutáveis;(ii) enorme fluxo de dados; (iii) orientados para o tempo (contém data e hora); (iv) baixa consistência. Como no-sql prioriza a escalabilidade e o desempenho, de forma a responder a estas características a melhor opção a utilizar seria uma base de dados não relacional com recurso a documentos.

O Cosmos BD é uma base de dados totalmente distribuída que suporta modelos de dados como documentos, gráficos e *key-value*. Para além disso, é possível criar, consultar e gerir recursos através da API SQL via REST. Quando é realizado um pedido HTTP para obter informação é necessário ter em conta os determinados cabeçalhos: (i) autorização; (ii) tipo de conteúdo; (iii) data; (iv) token de sessão; A autorização é um *token* de autorização para obter o pedido, podendo ser a chave primária como foi descrito no capítulo 5.5 (Segurança). O tipo de conteúdo indica o tipo de pedido HTTP (GET, POST, etc) que foi solicitado à base de dados. A data do pedido tem de ser RFC 1123 e o *token* de sessão tem o objetivo de garantir a consistência ao nível da sessão.

O AzureData é um SDK que suporta a API SQL em que abstrai todas as características do cabeçalho HTTP para o Azure facilitando o uso de *queries* à base de dados.

```
Query query = Query.Companion.select()
        .from(_equipment)
        .where( property: "deviceId", _idDevice);

AzureData.queryDocuments(_equipment, databaseId: "valuesDatabase", query, DictionaryDocument.class, maxPerPage: null,
        onCallback( response  -> {
    Log.e(TAG, msg: "Document list result: " + response.isSuccessful());
```

Figura 6.7- função que executa uma query para o Azure.

Na Figura 6.7 podemos observar na classe *QRCodeActivity* os campos necessários a preencher no corpo da *query*, onde é necessário fornecer o id do equipamento e posteriormente o pedido HTTP é enviado

para o Azure. Quando existe a necessidade de obter informação relevante aos equipamentos ou aos relatórios de manutenção é feito um pedido ao Azure onde a resposta a esse pedido segue com o corpo da resposta e o código de estado HTTP (código:200 caso seja bem-sucedido).
A base de dados está estruturada através de coleções e documentos em que cada documento contém a seguinte estrutura:

```
{
"id ": "String",
"deviceId ": "String",
"alert ": "String",
"value": "String",
"data": "String",
"rid": "String",
"self": "String",
"etag": "String",
"ts": Number
}
```

Apenas os campos *deviceId* e o *alert* são informações provenientes dos dispositivos e os restantes campos são gerados pela função do Azure, onde o id é uma função de *hash* da data de registo de forma a evitar que haja outro registo igual.

6.2 Realidade aumentada

6.2.1 Objetos 3D

Um *Renderable* é um modelo 3D que pode ser colocado em qualquer lugar tendo em conta os materiais e texturas. O *Sceneform* é uma API que facilita o *rendering* de objetos 3D em AR. Uma vez instalado no *ArCore* permite importar, visualizar e contruir recursos 3D numa aplicação AR no Android Studio. Foi escolhido esta API porque está integrada no *ArCore* e quando é necessário criar uma cena o programador apenas tem de criar um *renderable* a partir do objeto com o formato *.obj. A forma como a cena é estruturada é em forma de árvore que contém nós onde os objetos são *rendered*. Cada nó contém todas as informações que a API precisa para o posicionar o objeto 3D, nomeadamente a sua posição e orientação. Os nós também podem ser associados a outros nós, estabelecendo uma hierarquia pai-filho. Quando um nó pai se move ou a sua escala aumenta, os nós associados ao pai também aumentam de forma a garantir uniformidade entre os objetos 3D. O *Sceneform* também permite importar modelos com animações. Através da API do *Sceneform* é possível reproduzir e controlar a própria animação e anexar nós de um modelo. Essas animações são criadas num *software* de modelação de animações e posteriormente importados como um ficheiro *.sfb de forma a ser importados pelo *Sceneform*. Podemos observar na Figura 6.8 um objeto com um formato 3D após ser importado pelo *Sceneform*, para ser posteriormente usado pelo *ArCore*.

Figura 6.8- Formato de um modelo 3D em Android

6.3 Estrutura de aplicação

Um dos principios mais importante que precisa de ser seguido quando estamos a desenvolver uma aplicação/programa é a separação de conceitos, ou seja, separar em as classes ou funcionalidades consoante o seu propósito. No caso de uma aplicação móvel é um erro escrever o código todo numa só classe e não distinguir entre funcionalidades, ou seguindo um padrão arquitetural, por camadas. No capítulo 5.3 foi evidenciado o tipo de padrão arquitetural que serviu como modelo para a contrução da aplicação.

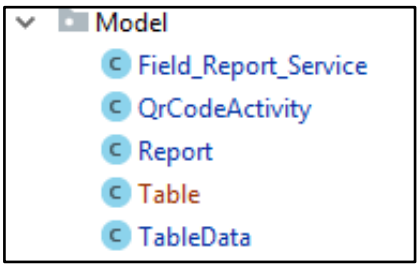

Figura 6.9 - *Model* da aplicação móvel

O *package Model* representado na Figura 6.*9* é responsável pela comunicação com a base de dados e pela representação de diversos tipos objetos. Classes como *Field_Report_Service* e *QrCodeActivity*, são responsável pela execução de pedidos de informação à base de dados. Quando é necessário gerar um novo tipo de relatórios é a classe *Field_Report_Service*, que realiza esses pedidos. Neste *package* também temos a representação conceptual de objetos.

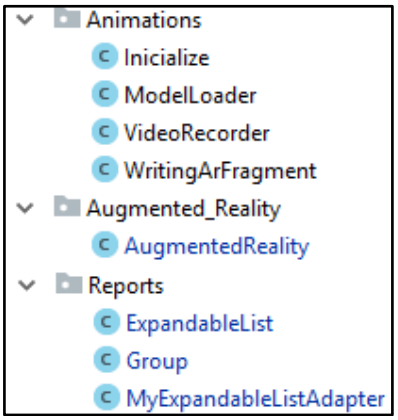

Figura 6.10- *Controllers* da aplicação móvel

Os *packages* representados na Figura 6.10, são baseados no padrão MVC como *controllers*. As classes representadas representam as funcionalidades principais da aplicação. O *package Animations* contém as classes que são responsáveis por reproduzir as respetivas animações no processo de desmontagem da máquina. O *package Augmented_Reality* contém uma única classe responsável por importar o modelo 3D da máquina e mostrar os seus problemas associados. O *package Reports* permite gerar relatórios de manutenção automaticamente após a resolução de um problema ou consultar relatórios anteriormente gerados

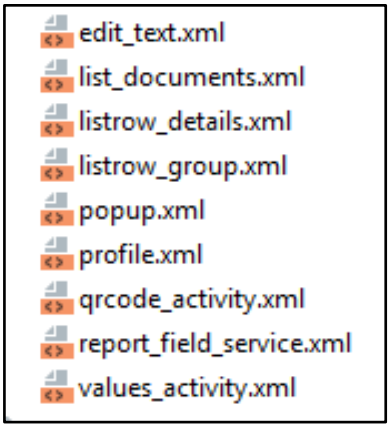

Figura 6.11 - *Views* da aplicação móvel

Na Figura 6.11 apenas estão representadas algumas das *Views* responsáveis pela interface do utilizador. Por exemplo, após receber a informação proveniente da base de dados sobre os problemas na máquina, a interface *values_activity.xml* apresenta sobre forma de tabelas esses mesmos valores ao utilizador.

6.4 Funcionalidades

Este subcapítulo descreve as diversas funcionalidades da aplicação desenvolvida. Sabendo que a vertente da aplicação foi desenvolvida numa vertente do técnico, o seu desenvolvimento tem como objetivo auxiliar as tarefas de manutenção ou reparação de equipamentos. Todas as funcionalidades foram discutidas nas reuniões de levantamento de requisitos de modo a perceber quais seriam as fraquezas que a aplicação poderia dar resposta.

Autenticação do utilizador:

A autenticação é o processo para determinar se um utilizador pode ou não aceder a um determinado sistema ou recurso, fornecendo acesso a sistemas após a verificação das suas credenciais. Para além disso, permite proteger as organizações evitando que utilizadores não autorizados tenham acesso aos recursos e dados confidenciais. Quando a aplicação é iniciada, a primeira atividade é constituída pela

autenticação do técnico pois, para utilizar a aplicação móvel, é necessário que um técnico esteja autenticado com uma conta Microsoft. Na Figura 6.12, é mostrado o botão para iniciar a autenticação e seguidamente, na Figura 6.13, é apresentada uma página web para validar a autenticação. Podemos também observar na mesma figura que existe um técnico com a sessão iniciada. Caso a resposta ao serviço de autenticação seja negativa, não será iniciada a aplicação e o técnico não consegue ter acesso aos recursos que precisa. Esta é uma funcionalidade inicial que previne os utilizadores não autorizados de ter acesso aos recursos da aplicação

Figura 6.12- Atividade inicial da aplicação móvel

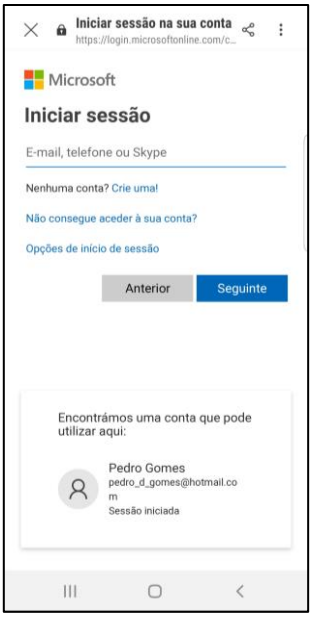

Figura 6.13- Login do utilizador

Perfil:

Quando a autenticação é validada é necessário obter as informações pessoais do técnico, sendo que o método responsável por isso é o *callGraphAPI()* da classe *MainActivity*. Esse método permite aceder ao Microsoft Graph que contém uma API REST para aceder a dados dos serviços Microsoft, permitindo assim obter informação pessoal do técnico. Após isso, o perfil do técnico é apresentado depois da sua autenticação onde é possível ver a sua página pessoal. A informação que é apresentada no perfil do técnico é com base na informação disponível na conta da Microsoft. Quando é feito um pedido HTTP para saber a informação pessoal do técnico, é fornecido por padrão um conjunto limitado de dados (p.ex. nome, id do técnico, trabalho, telefone, email e localização). No entanto, como podemos observar na Figura 6.14, a informação relevante é apenas o seu nome, id do técnico, o mail e o seu número, para posteriormente essa informação ser utilizada quando gerar um relatório de manutenção.

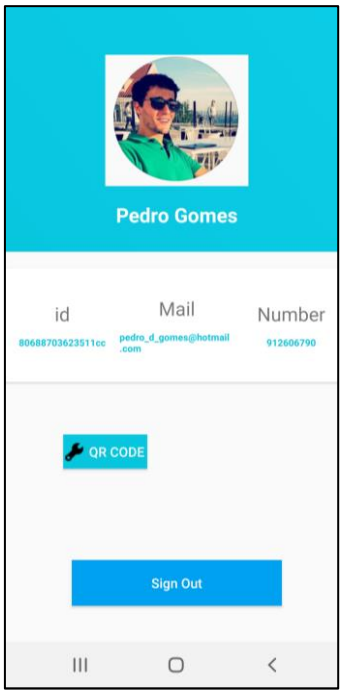

Figura 6.14- Perfil do utilizador

Realidade Aumentada:

Esta funcionalidade tem o objetivo de mostrar ao técnico o equipamento 3D que necessita de manutenção ou reparação. Quando o técnico faz a leitura do *QR Code* (Figura 6.15) presente no equipamento, esse código contém uma *string* correspondente ao seu id e posteriormente é gerado *na classe do QrCodeActivity* uma *query* para o Cosmos DB de forma a obter a informação especifica do equipamento. A informação proveniente da base de dados, as anomalias, são guardadas numa lista para posteriormente serem mostradas juntamente com o objeto 3D.

Após a verificação do *QR Code*, é iniciada a atividade de AR, sendo primeiramente necessário detetar a superfície onde irá ser colocado o objeto 3D. O objeto é colocado numa superfície para que o técnico consiga visualizar o equipamento na totalidade permitindo além disso, que o técnico possa mudar de localização e o mesmo continue no local. Este processo só é possível devido a API do *ArCore* para fixar objetos virtuais relativamente à sua posição no mundo real. Através dos métodos *ModelRenderable* e *ViewRenderable* disponibilizados pela API é possível criar modelos 3D do equipamento e processar modelos 2D no espaço 3D, respetivamente. Esta API deteta pontos característicos numa superfície ou contrastes entre superfícies antes colocar o objeto virtual, contudo, se esses pontos forem mínimos pode demorar muito tempo na deteção de planos e no pior caso não ser possível colocar um objeto numa determinada localização.

Como podemos observar na Figura 6.16 e Figura 6.17, é possível ver o objeto 3D de vários ângulos, possibilitando o técnico de visualizar detalhes caso tenha dificuldades em perceber onde está o problema.

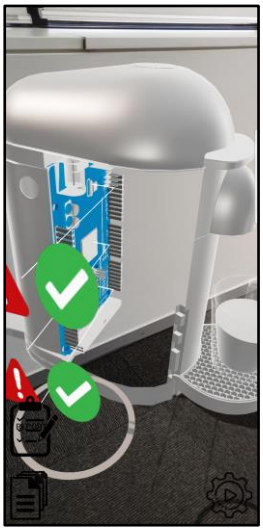

Figura 6.17- Máquina num ângulo diferente

Figura 6.16- Máquina a reparar

Figura 6.15- *Qr Code* da máquina

AR e IoT:

Uma das características fortes da aplicação é o facto de conseguir integrar AR com IoT. Esta integração permite que o técnico consiga visualizar não só o objeto 3D, mas também o tipo de anomalias que surgiram, conseguindo reduzir o tempo e o custo de reparação. Esta funcionalidade consiste em permitir que o técnico possa ter acesso aos recursos de AR e à informação de IoT em simultâneo.

Após a informação proveniente da *query* ao Cosmos DB realizada na classe *QrCodeActivity*, é apresentado o objeto 3D (Figura 6.19) e dois símbolos, um de perigo, que representa as anomalias existentes e um símbolo de correto que representa a resolução do problema, após o técnico realizar a reparação. Ambos os símbolos são nós anexados relativamente à posição do equipamento, caso haja alteração da posição do equipamento, a posição dos nós é ajustada.

Sempre que um técnico clica no símbolo de perigo aparecem os documentos guardados na base de dados, nomeadamente descrição do problema, os seus valores correspondentes e a data do problema Figura 6.18. Isso faz com que o técnico também consiga perceber algum padrão entre os problemas que surgiram, analisando por exemplo as horas da anomalia.

Quando o problema é resolvido substitui-se o símbolo de perigo pelo símbolo que representa o correto, permitindo assim guardar a informação da resolução do problema para gerar um relatório de manutenção. Para a substituição dos símbolos é necessário que haja a modificação dos nós que contém os *renderables*, removendo os nós que representam o símbolo de perigo.

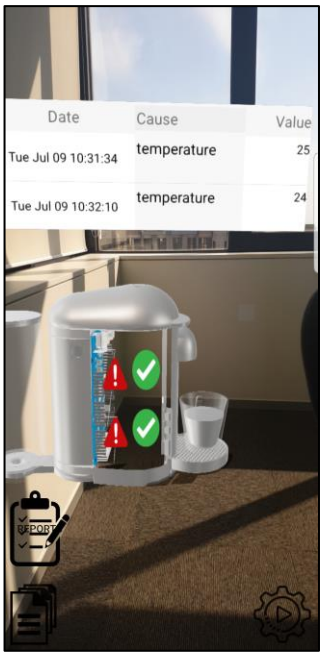

Figura 6.19- Tabelas com problemas da máquina

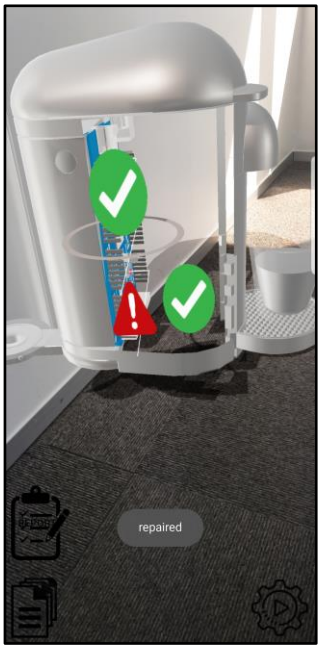

Figura 6.18- Máquina após a reparação de um problema

Relatórios de manutenção:

Os relatórios de manutenção têm como objetivo perceber quais foram os problemas que surgiram anteriormente num determinado equipamento e num determinado período.

Esta funcionalidade permite visualizar relatórios antigos ou gerar novos caso seja necessário. Quando um técnico pretende visualizar se existiram problemas recorrentes em algum sensor, pode aceder através da aplicação móvel aos relatórios de manutenção antigos que estão armazenados no Cosmos DB e perceber a origem do problema. Após essa verificação, é possível visualizar se o problema numa determinada peça é recorrente, ou se é a primeira vez que existe a anomalia. Os relatórios estão divididos em 4 tipos: (i)*repair*, (ii) *maintenance*, (iii) *installation*, (iv) *other*. Como podemos observar, na Figura 6.20, quando um técnico acede a lista de relatórios, pode escolher consoante o tipo e data. Por outro lado, o técnico pode gerar um relatório quando finaliza uma reparação, manutenção ou instalação.

Quando o técnico clica no símbolo de correto, é despoletado um evento no método *addProblems* da classe *AugmentedReality* que adiciona os problemas resolvidos numa lista. Assim que o técnico gerar um relatório de manutenção, apenas precisa de fornecer a descrição de como resolver o problema, pois os restantes campos são preenchidos automaticamente (Figura 6.21).

Os relatórios de manutenção são uma funcionalidade importante da aplicação pois através de informação anteriormente gerada é possível determinar padrões em anomalias e consequentemente aumentar a eficiência na deteção de problemas.

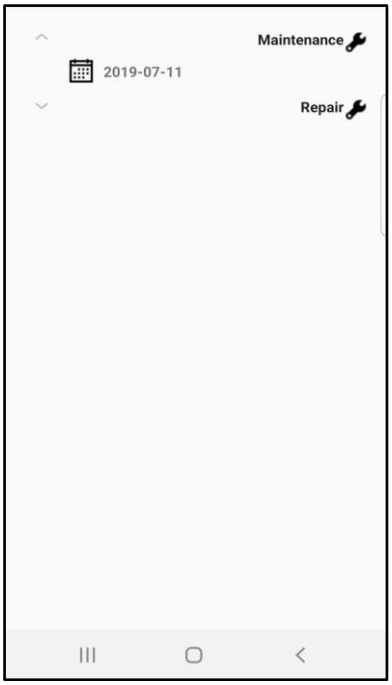

Figura 6.20 - Relatórios anteriormente gerados

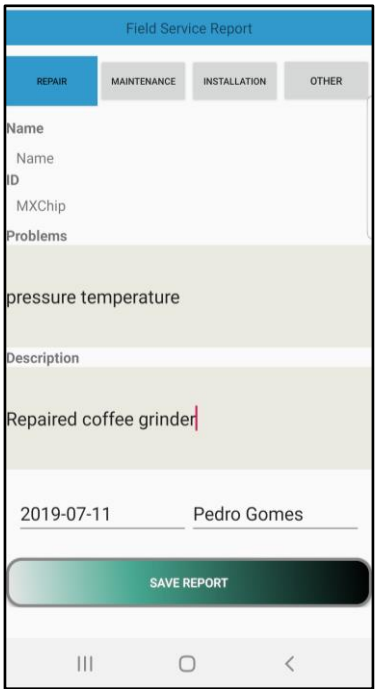

Figura 6.21- Exemplo de um relatório

Instruções de manutenção:

Nos serviços de reparação de equipamentos, um dos grandes desafios trata-se de ensinar a novos técnicos como operar um equipamento. Antigamente os técnicos precisavam de participar em cursos de treino ou ler manuais sobre a estrutura de um equipamento, contudo com a introdução de AR esse processo tem vindo a ser inovado. AR pode fornecer um treino prático, pois os técnicos podem receber instruções visuais sobre a máquina que estão a reparar, acelerando a fase de treino utilizando instruções 3D.

Esta funcionalidade tem como objetivo ajudar na reparação do equipamento utilizando instruções baseadas em AR. A Figura 6.*22* é um exemplo de uma máquina de café a ser reparada através de instruções 3D.

Figura 6.22- Instruções de reparação

O *SceneForm* permite importar modelos com animações, de forma a ser possível visualizar essa animação usando AR. Essas animações são importadas para um ficheiro no formato de *SceneForm Asset Definition* (*.sfa) e posteriormente o método *ModelRenderable* permite gerar objetos virtuais com as respetivas animações. As animações no ficheiro *.sfa contém um *array* de *clips* de animações (Figura 6.23) por ordem sequencial das animações. Contudo ao importar para o Sceneform, pode condicionar a forma como as mesmas são apresentadas. Por vezes, os *clips* das animações podem ser trocados, o que pode dificultar o processo de reparação.

```
animations: [
  {
    clips: [
      {
        name: 'screw_cross_hq.001|screw_cross_hq.001Action.004',
      },
      {
        name: 'nespresso_vertuo_plus.001|nespresso_vertuo_plus.001Action.003',
      },
      {
        name: 'unnamed3|unnamed3Action.004',
      },
    ],
    path: 'sampledata/assemblymachine62.fbx',
```

Figura 6.23- *Clips* de animações 3D

6.5 Principais obstáculos

Os problemas que vão ser descritos de seguida, foram encontrados durante o processo de desenvolvimento. Apesar de ter sido realizado um estudo prévio sobre que tecnologias e material a usar, foi difícil de prever que determinados problemas poderiam surgir devido a falta de experiência do

estagiário. Essa falta de experiência juntamente com o desconhecimento das tecnologias, originou que existisse problemas inesperados nas funcionalidades.

- Limitação do Sceneform:
  ArCore utiliza o Sceneform como *plug-in* para importar modelos 3D, contudo é um SDK criado recentemente pois a sua data de lançamento foi em 2018 e como qualquer tecnologia nova existem problemas que ainda não foram descobertos. No desenvolvimento do projeto uma das principais limitações associadas a esse *plug-in* foi a importação dos objetos 3D com animações. Quando uma animação é importada para a aplicação, no ficheiro *.sfa existe um *array* de animações que consoante a escolhe do utilizador a animação é reproduzida no objeto virtual. No entanto, ao importar os modelos, o Sceneform desordena essa animação fazendo com que fique confuso de perceber quais são as animações que é suposta visualizar. Para contornar isso, foi necessário percorrer o ficheiro *.sfa como representado na Figura 6.23 e reorganizar essas animações.

- Localização dos problemas da máquina:
  Uma das funcionalidades implementadas no projeto foi a capacidade de localizar um problema através de um formato 3D, ou seja, perceber numa máquina onde o problema surgiu. No entanto, apenas é possível ter acesso diretamente à estrutura do objeto quando utilizamos um *software* de modelação, que neste caso é o Blender. Por exemplo, na Figura 6.16 são mostrados dois símbolos de perigo, indicando onde está o problema num determinado sensor, no entanto, esses símbolos foram colocados consoante a posição do nó do objeto que contém a máquina e não do sítio exato onde está o problema. Ou seja, é necessário no Blender colocar um indicador, um traço e depois caso exista algum problema através do ArCore posicionar o símbolo consoante a posição do nó.

Os problemas descritos apenas foram possíveis de detetar enquanto decorria o desenvolvimento do projeto, devido as especificações das funcionalidades.

Capítulo 7 Testes

Nas várias fases de desenvolvimento de software é necessário considerar os a fase de testes como um processo fundamental para credibilidade de um produto. Por mais experiência que um programador tenha, é sempre necessário verificar se as funcionalidades desenvolvidas respondem como era esperado. Segundo [67] , podemos segmentar a fase de testes em 4, a fase onde percebemos como o software deve interagir com seu ambiente, a fase onde selecionamos os casos de teste, como será feita a execução e avaliação dos testes e a última fase de medição dos progressos dos testes.
Para a realização de testes foram executados testes de usabilidade, testes de compatibilidade. Para além disso, foram realizados também testes mais específicos na componente de AR e na componente de IoT.

7.1 Testes de usabilidade

Segundo [68], a usabilidade é definida como a facilidade de uso e o quanto um sistema é aceitável para um determinado grupo de utilizadores que realizam tarefas especificas. É fundamental estes dois conceitos, pois a facilidade com que um utilizador usa o sistema pode afetar o seu desempenho e consequentemente a satisfação de que utiliza, enquanto que a aceitação de um sistema afeta o quanto ele é usado.
Quando são realizados testes de usabilidade é necessário considerar 5 características: (i)capacidade de aprendizagem; (ii) eficiência; (iii) capacidade de memorizar; (iv) baixa taxa de erro; (v) satisfação. A capacidade de aprendizagem é o tempo que um utilizador demora até começar a usar rapidamente o sistema. A eficiência é a capacidade de usar o sistema com um elevado nível de produtividade. A capacidade de memorizar permite que um utilizador após algum tempo sem utilizar o sistema volte a trabalhar sem ter a necessidade de aprender. A baixa taxa de erro reflete-se na capacidade de trabalhar com o sistema sem efetuar erros e a satisfação torna o sistema fácil de usar.

Testes:
O contexto do projeto está direcionado para a vertente de um técnico, logo um técnico antes de usar uma aplicação receberá instruções de como usar a aplicação, tendo sido selecionados 5 indivíduos com algum conhecimento sobre o funcionamento da aplicação. Posto isto, apenas foram consideradas algumas das características acima mencionadas, tais como, a capacidade de aprendizagem, eficiência, baixa taxa de erro, por isso considerado como importante perceber se a aplicação preenchia esses requisitos.
Os 5 utilizadores foram sujeitos a tarefas baseadas nas funcionalidades implementadas no projeto (Tabela 7.1).

Id	Tarefas
Tr1	Visualizar a máquina em 3D
Tr2	Ver os problemas relacionados com a máquina
Tr3	Visualizar os relatórios anteriormente gerados
Tr4	Resolver os problemas da máquina
Tr5	Gerar um novo relatório de manutenção
Tr6	Seguir instruções de montagem/desmontagem da máquina

Tabela 7.1- Tarefas a realizar

Na Tabela 7.2 estão descritas de forma lógica como devem ser executadas as tarefas. Caso exista algum procedimento que não seja feito consoante o que está descrito na tabela deve ser anotado de forma a perceber o problema.

Id	Execução esperada
Tr1	Após o utilizador estar no seu perfil, faz a leitura do *QR Code* é direcionado para uma atividade que contém uma tabela com os problemas da máquina. Seguidamente carrega no botão "*Augmented Reality*" e tem de detetar uma superfície para colocar a máquina em 3D.
Tr2	Após colocar a máquina 3D numa superfície o utilizador consegue visualizar quantos problemas existem, e de seguida clica no símbolo de perigo para ver os problemas respetivos
Tr3	Na atividade de AR o utilizador tem de clicar no botão que indica *reports* de forma a ser capaz de visualizar a lista com os relatórios
Tr4	Um utilizador na atividade de AR carrega no símbolo do correto para o sistema registar que o problema foi resolvido.
Tr5	Após a correção dos problemas, o utilizador abre a atividade de gerar um relatório e fornece a descrição de como resolveu o problema e o seu tipo.
Tr6	O utilizador na atividade de AR, carrega no botão das animações e seguidamente seleciona no ícone onde dá início as animações

Tabela 7.2- Modo de execução das tarefas

Avaliação:

Durante a realização de cada tarefa da Tabela 7.1, foi pedido ao utilizador que avalia-se o grau de dificuldade de cada tarefa, enumerando de 1 até 3, sendo que 1(fácil), 2(médio) e 3(difícil).

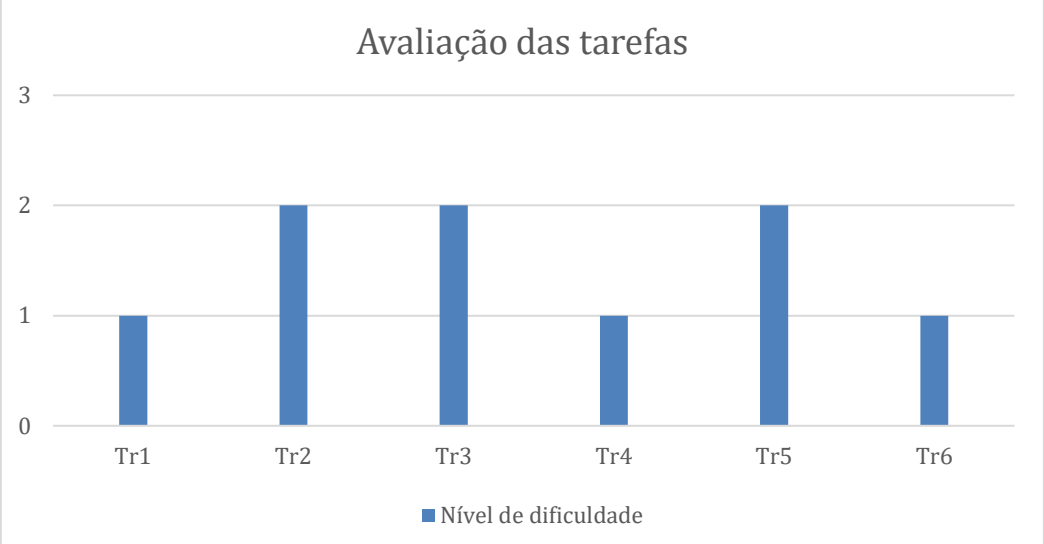

Gráfico 7.1- Níveis de dificuldade de cada tarefa

Observado o Gráfico 7.1, no geral os níveis de dificuldade 1 e 2 prevaleceram na avaliação das tarefas, concluindo que não houve grandes dificuldades na realização das mesmas. As tarefas 2, 3 e 5 causaram uma maior dificuldade pois exigiu mais tempo ao utilizador. Essa dificuldade é causada pela semelhança ou falta de informação nos símbolos presentes na máquina 3D. No entanto, não é um problema que impeça de realizar a tarefa, logo ser considerado nível 2. Na terceira tarefa , 3 utilizadores reportaram

que demoraram algum tempo a distinguir entre o botão de relatórios antigos e o de gerar um relatório, sugerindo tornar mais evidente essa diferença.
Concluindo, não houve nenhuma dificuldade de nível 3, podendo assim afirmar que a aplicação respeita as características acima propostas, capacidade de aprendizagem, eficiência e baixa taxa de erro.

Sugestões:

Apesar do nível de dificuldade ser baixo, ou seja, os utilizadores realizaram as tarefas com sucesso, forneceram sugestões, nomeadamente:

- Tr2: Colocar um indicador sobre o símbolo de perigo para ser mais intuitivo ver os problemas que a máquina tem.

-Tr3: Distinguir melhor o botão de gerar um relatório ou ver os antigos, pois existe dificuldade em distinguir a diferença entre os mesmos.

-Tr6: Colocar as instruções que aparecem nos *popup's* ligeiramente maior.

-Tr7: Receber informação quando carrega no botão de guardar base de dados.

7.2 Testes de compatibilidade

Os testes de compatibilidade é um dos vários tipos de testes de software executados num sistema, com o objetivo de garantir a compatibilidade em outros dispositivos. Neste caso é com o objetivo de verificar o desempenho em diferentes dispositivos. É fundamental a realização deste teste devido a existência de múltiplos dispositivos, contudo devido aos requisitos impostos pelo *ArCore* apenas consideramos dipositivos que tenham a versão de Android superior a 7.0 e uma lista especifica de dispositivos. A aplicação foi elaborada para a vertente do técnico, e como na Europa as empresas que prestam serviços de reparação e manutenção utilizam maioritariamente Android, apenas foi testado esse sistema operativo.

Dispositivo	**Sistema Operativo**	**Especificações**
Samsung S9	Android 9.0	Qualcomm SDM845 Snapdragon 845 (10 nm), 4GB RAM
Xiaomi pocophone x1	Android 9.0	Qualcomm SDM845 Snapdragon 845 (10 nm), 6GB RAM
Huawei Mate 20 lite	Android 8.0	HiSilicon Kirin 980 (7 nm), 4GB RAM
Huawei p20	Android 8.1	Hisilicon Kirin 970 (10 nm), 4 GB RAM

Tabela 7.3 - Dispositivos móveis utilizados

Após os testes de compatibilidade nos dispositivos apresentados na Tabela 7.3, foi possível tirar conclusões relativas a forma como a interface do utilizador se ajustava consoante os diferentes ecrãs. Sendo que após a deteção desse problema foram feito ajustes no posicionamento nas *Views*, de forma a que a aplicação se ajustasse consoante os dispositivos. Na Figura 7.1 e na Figura 7.2 estão representados dois dos 4 dispositivos móveis utilizados para testes de compatibilidade

Figura 7.2- Samsung S9


Figura 7.1- Huawei mate 20 lite

7.3 Testes de realidade aumentada

Os testes de AR são mais demorados e menos consistentes que os testes tradicionais de *software*. É importante em AR perceber quais são os ambientes corretos para testar uma aplicação e com objetos diferentes, em cenas com vários tipos de iluminação. Quando se trata de uma aplicação AR também é importante ter em conta a capacidade de detetar e compreender o ambiente do mudo real, ou seja, detetar pontos característicos num plano. O *ArCore* procura *clusters* de pontos que estão em superfícies horizontais ou verticais comuns, de forma a tornar possível colocar um objeto virtual numa dessas superfícies. Isso por vezes pode ser uma limitação devido ao pouco contraste nas superfícies, pois dificulta a deteção desses pontos.

Um dos testes que realizado tem o objetivo de perceber como uma superfície com muito ou pouco contraste pode influenciar a aplicação. Para isso, foram selecionados diferentes ambientes e superfícies de forma a analisar o tempo de resposta da aplicação na deteção das mesmas. Para calcular o tempo de deteção foi implementado um contador na aplicação. O contador mostra a diferença de tempo em milissegundos, desde que a atividade é iniciada até ao momento da deteção. No pior caso vamos considerar uma superfície totalmente branca, e nos outros casos e uma superfície com contraste.

- **Superfície Branca:**
  Tempo de deteção de superfície 11s e 8s

Figura 7.3- Tempo de deteção da superfície, 8 segundos

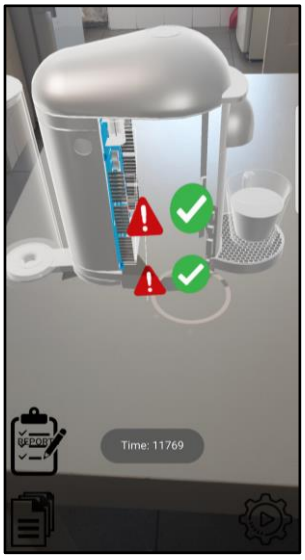

Figura 7.4- Tempo de deteção da superfície, 11 segundos

- **Superfície com contrastes:**
  Tempo de deteção de superfície 5s e 4s

Figura 7.5- Tempo de deteção da superfície, 5 segundos

Figura 7.6- Tempo de deteção da superfície, 4 segundos

Analisando o tempo que cada superfície demora a ser detetada podemos concluir que, quando não existe contraste, ou seja, pontos que permitam detetar rapidamente, o tempo aumenta aproximadamente para o dobro. No entanto, quando existe limites nas superfícies ou texturas o tempo de deteção é reduzido. No geral, o tempo que a aplicação demora a detetar uma superfície é baixa, o que é um bom indicador. A Figura 7.4 representa o pior caso, onde não existem contrastes, demora 11 segundos a detetar uma superfície, o que pode ser bastante tempo, pois nos casos de usabilidade testados ao fim de 6/7 segundos os utilizadores pensaram que existiria algum problema. Na Figura 7.6 está representado o melhor caso, onde o tempo fio apenas de 4 segundo pois existem muitos pontos característicos que permitem detetar facilmente as superfícies.

7.4 Testes do sistema IoT

Em geral, IoT é uma rede de dispositivos de rápido crescimento que recolhe grande quantidade de dados. A plataforma IoT Azure hub contêm uma infraestrutura *end-to-end* para permitir a troca de informação e a gestão de dispositivos em tempo real. Cada IoT *hub* contém conjunto de pontos de extremidade para o seu *back*-end da solução que permite a comunicação M2M. Quando um dispositivo envia informação para a nuvem, utiliza um ponto de extremidade fornecido pelo sistema para facilitar a comunicação entre o dispositivo e o *hub*.
Segundo [69], o IoT *hub* está preparado para ter milhares de dispositivos conectados simultaneamente, conseguindo assim responder a uma característica importante, isto é, a escalabilidade do sistema. Para garantir que um sistema funciona corretamente é necessário testar características como desempenho e heterogeneidade fazendo uma avaliação de desempenho com diferentes tipos de dispositivos. Um teste de desempenho tem como objetivo perceber quantas conexões simultâneas um o *hub* consegue dar resposta. Para isso foram simulados 21 dispositivos, onde 5 estão num Raspberry Pi, 15 num computador através de um script em python e o MxChip. Os Raspberry Pi envia mensagens a cada 5 segundos, o computador enviam mensagens a cada segundo, enquanto que o MXChip envia a cada minuto

- Simulação 1: No Raspberry Pi foram simulados 5 dispositivos a enviar mensagens durante aproximadamente 5 minutos o que poderíamos calcular que chegariam ao IoT *hub* aproximadamente 300 mensagens. Como podemos observar na Gráfico 7.2, foram recebidas 297 mensagens, concluindo assim que a resposta do sistema ao aumento do número de dispositivos não afeta a quantidade de mensagens que o sistema deve tratar

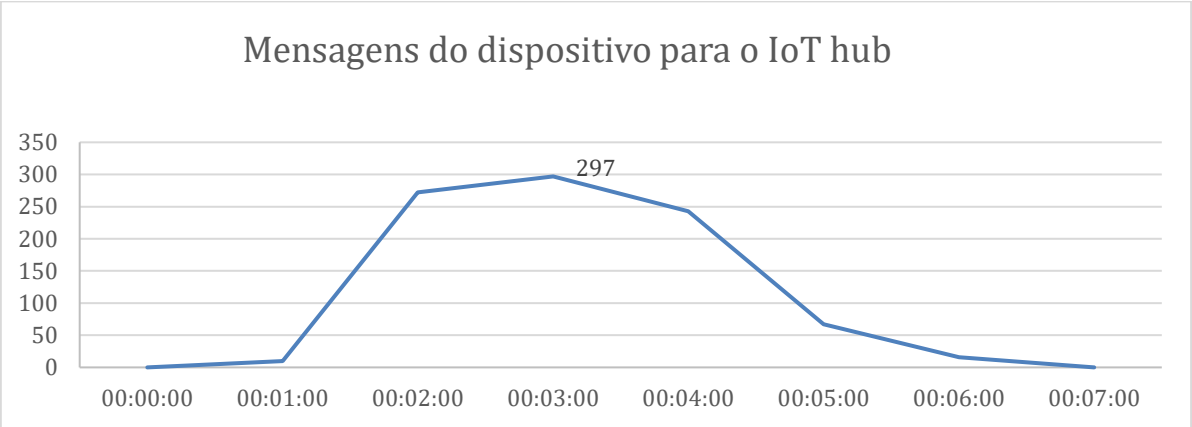

Gráfico 7.2- Número de mensagens enviadas por 5 dispositivos

- Simulação 2: Na segunda simulação houve um aumento da carga com 15 dispositivos a enviar dados a cada segundo, durante aproximadamente 1 minuto. Calculando as mensagens enviadas nesse período de tempo, podemos obter um intervalo entre 600 a 700 mensagens que deveriam chegar ao IoT *hub*. Como podemos observar na Gráfico 7.3 foram recebidas 692 mensagens, concluindo assim que apesar do número de dispositivos aumentar o sistema consegue dar resposta sem problemas.

Gráfico 7.3- Número de mensagens enviadas por 15 dispositivos

Capítulo 8 Conclusões e trabalho futuro

8.1 Conclusões

Atualmente existem cada vez mais organizações que começam a adotar IoT com recurso a AR. A combinação dessas duas tecnologias já não é algo surpreendente, uma vez que permite a criação de valor, ou seja, novos produtos e otimização de serviços. Alguma das principais preocupações das empresas inclui reduzir custos e tempo, pois só assim é possível um serviço conseguir ser competitivo perante o mercado. Embora IoT, juntamente com o poder da nuvem já tenha um grande impacto sobre serviços de reparação e manutenção, integrar AR é como um complemento para estender a informação e permitir uma interação de forma mais eficiente. Neste projeto foi desenvolvida uma aplicação em que o seu objetivo era integrar um sistema IoT com AR de forma a facilitar a interação de um técnico com o sistema e conseguir reduzir tempo e custos associados a uma reparação.

Inicialmente foi necessário perceber como simular um sistema IoT e para isso foi usado o MxChip como ferramenta de simulação, facilitando assim a integração com o sistema IoT da Microsoft. A utilização da nuvem do Azure, permitiu utilizar serviços de computação *serveless* como o de tratamento de dados, utilizar base de dados que armazenam documentos em forma de documentos e um serviço de autenticação que oferece recursos de verificação de identidade. O desenvolvimento da aplicação foi projetado na vertente de técnico de reparação, por isso, foi opção usar Android como sistema operativo. Para além disso, foi utilizado o ArCore como um SDK capaz de reproduzir modelos 3D. A utilização desse SDK permitiu reproduzir modelos virtuais num mundo real e acrescentar informação proveniente do sistema IoT sobre uma determinada máquina. A aplicação foi baseada no padrão arquitetural MVC, facilitando assim a perceção das funcionalidades e do código.

Foram feitos diversos testes a aplicação, testes de usabilidade e compatibilidade de forma a perceber quais os problemas associados à sua conceção, problemas que poderiam surgir e se não havia restrições entre diversos dispositivos móveis. Ainda na componente dos testes, foram também realizados testes mais específicos para AR e IoT. Após a realização dos testes de AR, foi possível perceber como os pontos característicos para detetar uma superfície podem influenciar o desempenho da aplicação. Na vertente de IoT foi testada a escalabilidade e interoperabilidade do sistema através de um aumento no número de mensagens num intervalo curto.

Considerando que os requisitos e os objetivos iniciais definidos foram cumpridos, foi possível construir uma aplicação que fornecesse apoio nas tarefas de manutenção de uma máquina. Como podemos observar nos testes realizados, a componente de IoT responde com eficiência ao aumento de carga e a componente de AR consegue reproduzir objetos 3D com grande detalhe. No entanto, a integração de ambas permitiu que fosse possível não só facilitar na visualização de problemas, mas também aumentar a rapidez com que um técnico realiza uma tarefa.

8.2 Reflexão critica

Considerando que o principal objetivo deste estágio seria desenvolver um projeto que proporciona-se ao estagiário aprender metodologias de trabalho, ferramentas novas e tomar decisões nomeadamente a tecnologias, pode-se concluir que o objetivo foi cumprido. Do ponto de vista académico, foi possível por em prática inúmeras tecnologias e linguagens de programação desenvolvidas ao longo do curso, aproveitando assim para aprender mais sobre as mesmas. Do ponto de

vista profissional foi possível aprender metodologias de trabalho e como desenvolver um projeto num ambiente empresarial.
Analisando o projeto e as tecnologias usadas e começando na componente IoT, o serviço de computação na nuvem Azure utilizada neste projeto, é bastante útil e permite que seja fácil de implementar um sistema IoT que utilize uma grande quantidade de dispositivos. Para além disso, permite serviços de autenticação de dispositivos sendo uma mais valia na componente da segurança. É importante ao desenvolver um sistema IoT considerar características como escalabilidade, heterogeneidade, entre outras, e o sistema de IoT do Azure permite lidar com essas características. Ao utilizar esses serviços da nuvem foi também possível aprender linguagens como C# e bibliotecas .NET, permitindo assim desenvolver em tecnologia Microsoft. Apesar de existir diversas opções de serviços da nuvem, o Azure contém serviços que oferecem um número substancial de serviços, desde armazenamento até a gestão de dispositivos. Inclui também um SDK que permite as aplicações móveis obterem informação da nuvem, facilitando assim o desenvolvimento de aplicações móveis que precisem serviços da nuvem. Para além disso, os serviços da Microsoft são muito utilizados no âmbito empresarial onde estava inserido, permitindo assim ter um suporte em caso de problemas.
Na componente de AR seria importante optar por um desenvolvimento nativo para obter todas as capacidades de um dispositivo, pois a reprodução e modelação 3D de objetos requer muita capacidade do dispositivo, nomeadamente dos seus componentes, bibliotecas e inúmeras extensões. Posto isso, e dada a variedade de dispositivos Android existentes no mercado, seria uma opção muito fiável desenvolver para esse sistema operativo.
Toda a aplicação desenvolvida, desde o sistema de IoT até a AR, permitiu entrar em contacto com diversas tecnologias e perceber como dois conceitos que separadamente podem não ter um valor tão vasto, mas que a sua integração permite reforçar o seu valor junto das organizações empresariais.

8.3 Trabalho futuro

No subcapítulo da arquitetura foi descrito uma arquitetura de 3 camadas onde a maior parte do domínio de informações e operações está situada na camada da plataforma da nuvem. Embora a nuvem forneça serviços como computação, armazenamento e comunicação, esses recursos quando centralizados podem causar problemas como latência e afetar a eficiência de um sistema. Considerando estes problemas, o seguimento do projeto passa por implementar computação na borda. A computação na borda é uma infraestrutura de computação descentralizada onde os recursos computacionais estão localizados na borda da rede. Este conceito não é apenas uma maneira de coletar dados antes de enviar para a nuvem, mas também um método de analisar e processar esses dados, “filtrando” apenas o que é necessário enviar. Como consequência seria possível evitar problemas relacionados com a rede, como por exemplo, atrasos na transmissão de dados. No âmbito deste projeto, o objetivo de implementar esse sistema, seria perceber como é que determinados padrões poderiam levar a uma anomalia analisando os dados em tempo real da máquina. Através dos dados recolhidos que anteriormente geraram anomalias, fazer uma análise preditiva e antes do problema acontecer agir sobre o mesmo de forma a evitar que uma máquina fique inutilizável.

Bibliografia

[1] M. Latif, Y. Lakhrissi, E. H. Nfaoui, and N. Es-Sbai, “Cross platform approach for mobile application development: A survey,” in *2016 International Conference on Information Technology for Organizations Development, IT4OD 2016*, 2016.

[2] “VR vs AR vs MR.” [Online]. Available: https://rubygarage.org/blog/difference-between-ar-vr-mr. [Accessed: 14-Dec-2018].

[3] “ArCore.” .

[4] P. Sethi and S. R. Sarangi, “Internet of Things: Architectures, Protocols, and Applications,” *Journal of Electrical and Computer Engineering*. 2017.

[5] V. Karagiannis, P. Chatzimisios, F. Vazquez-Gallego, and J. Alonso-Zarate, “A Survey on Application Layer Protocols for the Internet of Things,” *Trans. IoT Cloud Comput.*, 2015.

[6] Z. Rashid, J. Melià-Seguí, R. Pous, and E. Peig, “Using Augmented Reality and Internet of Things to improve accessibility of people with motor disabilities in the context of Smart Cities,” *Futur. Gener. Comput. Syst.*, 2017.

[7] A. Badouch, S. Krit, M. Kabrane, and K. Karimi, “Augmented Reality services implemented within Smart Cities, based on an Internet of Things Infrastructure, Concepts and Challenges,” 2018.

[8] “Number of smartphones sold to end users worldwide from 2007 to 2017 (in million units).” [Online]. Available: https://www.statista.com/statistics/263437/global-smartphone-sales-to-end-users-since-2007/. [Accessed: 13-Dec-2018].

[9] “Sobre a Accenture.” [Online]. Available: https://www.accenture.com/pt-pt/company. [Accessed: 10-Dec-2018].

[10] D. Amin and S. Govilkar, “Comparative Study of Augmented Reality Sdk’s,” *Int. J. Comput. Sci. Appl.*, 2015.

[11] “VuMark.” [Online]. Available: https://library.vuforia.com/articles/Training/VuMark. [Accessed: 29-Nov-2018].

[12] C. Duarte, “Aplicação de realidade aumentada mobile com desenhos de arquitetura na cidade do Porto,” 2017.

[13] K. Ćuković, Saša & Gattullo, Michele & Pankratz, Frieder & Devedzic, Goran & Carrabba, Ernesto & Baizid, “Marker Based vs. Natural Feature Tracking Augmented Reality Visualization of the 3D Foot Phantom,” 2015.

[14] “Cloud Anchor.” [Online]. Available: https://developers.google.com/ar/develop/java/cloud-anchors/quickstart-android. [Accessed: 18-Dec-2018].

[15] “ArKit.” [Online]. Available: https://developer.apple.com/arkit/. [Accessed: 28-Nov-2018].

[16] F. Herpich, R. L. M. Guarese, and L. M. R. Tarouco, “A Comparative Analysis of Augmented Reality Frameworks Aimed at the Development of Educational Applications,” *Creat. Educ.*, 2017.

[17] “Augmented Reality on Smartphones.” [Online]. Available: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=158311. [Accessed: 28-Nov-2018].

[18] “EasyAr.” [Online]. Available: https://www.easyar.com/doc/EasyAR SDK/Getting Started/Getting-Started-with-EasyAR.html#what-is-easyar-sdk. [Accessed: 29-Nov-2018].

[19] F. D. Miorandi, D. Sicari, S. Pellegrini and I. Chlamtac, “Ad Hoc Networks Internet of things: Vision, applications and research challenges,” *Ad Hoc Networks*, 2012.

[20] M. Wu, T. J. Lu, F. Y. Ling, J. Sun, and H. Y. Du, “Research on the architecture of Internet of Things,” in *ICACTE 2010 - 2010 3rd International Conference on Advanced Computer Theory and Engineering, Proceedings*, 2010.

[21] P. Mell and T. Grance, “The NIST definition of cloud computing - SP 800-145,” *NIST Spec. Publ.*, 2011.

[22] F. Bonomi, R. Milito, J. Zhu, and S. Addepalli, “Fog computing and its role in the internet of things,” in *Proceedings of the first edition of the MCC workshop on Mobile cloud computing - MCC ’12*, 2012.

[23] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The social internet of things (SIoT) - When social networks meet the internet of things: Concept, architecture and network characterization,” *Comput. Networks*, 2012.

[24] A. Čolaković and M. Hadžialić, “Internet of Things (IoT): A review of enabling technologies, challenges, and open research issues,” *Computer Networks*. 2018.

[25] “User Datagram Protocol.” [Online]. Available: https://searchnetworking.techtarget.com/definition/UDP-User-Datagram-Protocol. [Accessed: 20-Dec-2018].

[26] “What is SSL, TLS and HTTPS.” [Online]. Available: https://www.websecurity.symantec.com/security-topics/what-is-ssl-tls-https.

[Accessed: 20-Dec-2018].

[27] "AMQP is the Internet Protocol for Business." [Online]. Available: https://www.amqp.org/about/what. [Accessed: 30-Nov-2018].

[28] "The Web Application Messaging Protocol." [Online]. Available: https://wamp-proto.org/. [Accessed: 20-Dec-2018].

[29] P. R. Pietzuch and J. M. Bacon, "Hermes: A distributed event-based middleware architecture," in *Proceedings - International Conference on Distributed Computing Systems*, 2002.

[30] A. I. Wasserman, "Software engineering issues for mobile application development," 2010.

[31] "Approaching Mobile, Understanding the Three Ways to Build Mobile Apps." [Online]. Available: https://www.telerik.com/docs/default-source/whitepapers/choose-right-approach-mobile-app-developmentbb581d10116543e79a9febdb187fd0a3.pdf?sfvrsn=0. [Accessed: 30-Oct-2018].

[32] "Native vs. Cross-Platform: which reign in AR app development world?" [Online]. Available: https://invisible.toys/native-vs-cross-platform/. [Accessed: 15-Nov-2018].

[33] C. P. Rahul Raj and S. B. Tolety, "A study on approaches to build cross-platform mobile applications and criteria to select appropriate approach," in *2012 Annual IEEE India Conference, INDICON 2012*, 2012.

[34] M. Palmieri, I. Singh, and A. Cicchetti, "Comparison of cross-platform mobile development tools," in *2012 16th International Conference on Intelligence in Next Generation Networks, ICIN 2012*, 2012.

[35] "Difference between MVVM and MVP." [Online]. Available: http://www.differencebetween.net/technology/difference-between-mvvm-and-mvp/. [Accessed: 25-Jun-2019].

[36] "Model-View-Controller." [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff649643(v=pandp.10). [Accessed: 24-Jun-2019].

[37] T. Lou, "A comparison of Android Native App Architecture MVC, MVP and MVVM," Eindhoven University of Technology.

[38] "Implementing the MVVM Pattern Using the Prism Library for WPF." .

[39] "Data binding in depth." [Online]. Available: https://docs.microsoft.com/en-us/windows/uwp/data-binding/data-binding-in-depth. [Accessed: 30-Jun-2016].

[40] J. H. Christensen, "Using RESTful web-services and cloud computing to create next

generation mobile applications," 2009.

[41] "Simple Object Access Protocol (SOAP) 1.1." [Online]. Available: https://www.w3.org/TR/2000/NOTE-SOAP-20000508/. [Accessed: 20-Dec-2018].

[42] K. Wagh and R. Thool, "A Comparative Study of SOAP Vs REST Web Services Provisioning Techniques for Mobile Host," *J. Inf. Eng. Appl.*, 2012.

[43] "Conheça o Android Studio." [Online]. Available: https://developer.android.com/studio/intro/. [Accessed: 29-Nov-2018].

[44] "Xcode." [Online]. Available: https://developer.apple.com/xcode/. [Accessed: 30-Nov-2018].

[45] "ViroReact." [Online]. Available: https://docs.viromedia.com/docs/viro-platform-overview. [Accessed: 29-Nov-2018].

[46] S. Rautmare and D. M. Bhalerao, "MySQL and NoSQL database comparison for IoT application," in *2016 IEEE International Conference on Advances in Computer Applications, ICACA 2016*, 2017.

[47] N. Leavitt, "Will NoSQL Databases Live Up to Their Promise?," *Computer (Long. Beach. Calif).*, 2010.

[48] R. Cattell, "Scalable SQL and NoSQL data stores," *ACM SIGMOD Rec.*, 2011.

[49] A. Nayak, A. Poriya, and D. Poojary, "Type of NOSQL databases and its comparison with relational databases," *Int. J. Appl. Inf. Syst.*, 2013.

[50] A. M. De Souza, E. P. V Prado, V. Sun, and M. Fantinato, "Critérios para Seleção de SGBD NoSQL : o Ponto de Vista de Especialistas com base na Literatura," *Simpósio Bras. Sist. Informação*, 2014.

[51] J. S. Van Der Veen, B. Van Der Waaij, and R. J. Meijer, "Sensor data storage performance: SQL or NoSQL, physical or virtual," in *Proceedings - 2012 IEEE 5th International Conference on Cloud Computing, CLOUD 2012*, 2012.

[52] S. F. Leau, Y.B., Loo, W.K., Tham, W.Y. and Tan, "Software development life cycle AGILE vs traditional approaches," *Int. Conf. Inf. Netw. Technol.*, 2012.

[53] A. Highsmith, J. and Cockburn, "Agile software development: The business of innovation. Computer," pp. 120–127, 2001.

[54] K. Beck *et al.*, "Manifesto for Agile Software Development," *The Agile Alliance*, 2001. .

[55] T. Dybå and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," *Information and Software Technology*. 2008.

[56] J. Jetter, J. Eimecke, and A. Rese, "Augmented reality tools for industrial applications:

What are potential key performance indicators and who benefits?," *Comput. Human Behav.*, 2018.

[57] "WELCOMING AUGMENTED REALITY TO THE INDUSTRIAL WORLD." [Online]. Available: https://www.cat.com/en_US/articles/customer-stories/built-for-it/augmented-reality.html. [Accessed: 10-Dec-2018].

[58] M. A. Frigo, E. C. C. da Silva, and G. F. Barbosa, "Augmented Reality in Aerospace Manufacturing: A Review," *J. Ind. Intell. Inf.*, 2016.

[59] B. T. Holger Glockner, Kai Jannek, Johannes Mahn, "DHL." [Online]. Available: http://www.dhl.com/content/dam/downloads/g0/about_us/logistics_insights/csi_augmented_reality_report_290414.pdf. [Accessed: 10-Jan-2019].

[60] "ThingWorx, Augmented Reality, and IoT in the workspace." [Online]. Available: https://design-engine.com/thingworxaugmented-reality-and-iot-in-the-workplace/. [Accessed: 10-Jan-2019].

[61] G. B. Shi-Wan Lin (Thingswise/Intel), Bradford Miller (GE), Jacques Durand (Fujitsu) and B. M. (RTI) and M. C. (SAP) (IBM), Amine Chigani (GE), Robert Martin (MITRE), "The Industrial Internet of Things Volume G1: Reference Architecture." .

[62] "Blender." [Online]. Available: https://www.blender.org/about/. [Accessed: 04-Jul-2019].

[63] "Native apps." [Online]. Available: https://docs.microsoft.com/en-us/Azure/active-directory/develop/native-app#diagram. [Accessed: 01-Jul-2019].

[64] "Database Security." .

[65] "Secure access to data in Azure Cosmos DB." .

[66] "Azure Cosmos DB: REST API Reference." [Online]. Available: https://docs.microsoft.com/en-us/rest/api/cosmos-db/. [Accessed: 03-Jul-2019].

[67] J. A. Whittaker, "What is software testing? And why is it so hard?"

[68] A. Holzinger, "Usability engineering methods for software developers," *Commun. ACM*, 2005.

[69] "What is Azure IoT Hub?" [Online]. Available: https://docs.microsoft.com/en-us/Azure/iot-hub/about-iot-hub. [Accessed: 08-Jul-2019].